AD-A101 852 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 12/1
PRELIMINARY ANALYSIS OF A BREADTH-FIRST PARSING ALGORITHM: THEO--ETC(U)
JUN 81 W A MARTIN, K W CHURCH, R S PATIL N00014-75-C-0661
UNCLASSIFIED MIT/LCS/TR-261 NL

END
DATE FILMED
8-81
DTIC

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TR-261

PRELIMINARY ANALYSIS OF A BREADTH-FIRST PARSING ALGORITHM: THEORETICAL AND EXPERIMENTAL RESULTS

William A. Martin
Kenneth W. Church
Ramesh S. Patil

This research was supported (in part) by the National Institutes of Health Grant No. 1 PO1 LM 03574-02 from the National Library of Medicine, and by the Defense Advanced Projects Agency monitored by the Office of Naval Research under Contract No. N0001 4-75-C-0661.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MIT/LCS/TR-261	2. GOVT ACCESSION NO.: AD-A107 852
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results.	5. TYPE OF REPORT & PERIOD COVERED: June 1981
	6. PERFORMING ORG. REPORT NUMBER: MIT/LCS/TR-261
7. AUTHOR(s): William A. Martin, Kenneth W. Church, Ramesh S. Patil	8. CONTRACT OR GRANT NUMBER(s): DARPA N00014-75-C-0661, NR 1-P01-LM-03374-02
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, Mass. 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: DARPA, 1400 Wilson Blvd., Arlington, Va. 22209	12. REPORT DATE: June 1981
	13. NUMBER OF PAGES: 86
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Department of the Navy, Information Systems Program, Arlington, Va. 22217	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

This document is approved for public release and sale; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parsing, chart parsing, natural language processing, Earley's algorithm

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We will trace a brief history of context-free parsing algorithms and then describe some representation issues. The purpose of this paper is to share our philosophy and experience in adapting a well-known context free parsing algorithm (Earley's algorithm and variations thereof) to the parsing of a difficult and wide ranging corpus of sentences. The sentences were gathered by Malhotra in an experiment which fooled businessmen users into thinking they were interacting with a computer, when they were actually interacting with Malhotra in another room. The Malhotra corpus is considerably more difficult -

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

409648

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

than a second collection published by the LADDER Group. Both collections are given in the appendices. Section 4 compares empirical results obtained from these collections against theoretical predictions.

Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results

William A. Martin
Kenneth W. Church
Ramesh S. Patil

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

June 1981

Abstract

We will trace a brief history of context-free parsing algorithms and then describe some representation issues. The purpose of this paper is to share our philosophy and experience in adapting a well-known context free parsing algorithm (Earley's algorithm [8, 9] and variations thereof [28, 13, 26, 27]) to the parsing of a difficult and wide ranging corpus of sentences. The sentences were gathered by Malhotra [22] in an experiment which fooled businessmen users into thinking they were interacting with a computer, when they were actually interacting with Malhotra in another room. The sentences are given in Appendix I. The Malhotra corpus is considerably more difficult than a second collection given in Appendix II (originally published in [15]). Section 4 compares empirical results obtained from these collections against theoretical predictions.

Key Words: Parsing, Chart Parsing, Natural Language Processing, Earley's Algorithm

This research was supported (in part) by the National Institutes of Health Grant No. 1 P01 LM 03374-02 from the National Library of Medicine, and by the Defense Advanced Research Projects Agency (DOD) monitored by the Office of Naval Research under Contract No. N00014-75-C-0661.

CONTENTS

This paper will trace a brief history of context-free parsing algorithms and then describe some representation issues. Finally we will present a new parser (EQSP)[1] which has better average case performance because of its improved representation. The purpose of this paper is to share our philosophy and experience in adapting a well-known context free parsing algorithm (Earley's algorithm [8, 9] and variations thereof [28, 13, 26, 27]) to the parsing of a difficult and wide ranging corpus of sentences. The sentences were gathered by Malhotra [22] in an experiment which fooled businessmen users into thinking they were interacting with a computer, when they were actually interacting with Malhotra in another room. The sentences are given in Appendix I. The use of constructions and punctuation varies widely from user to user and contrasts sharply in its variety with the sentences in Appendix II, which originally appeared in [15]. These sentences, which we will call the LADDER-TODS Collection, were chosen to make a wide variety of constructions possible while limiting the difficulty of parsing. To move from the LADDER-TODS to the MALHOTRA corpus required an approximate doubling of our code.

We believe that syntax, semantics, and pragmatics are all required to parse a corpus like the MALHOTRA. However our view is that, computationally, syntactic constraints are in general the cheapest to apply, while semantics and pragmatics are progressively more expensive. Therefore, we have chosen to step aside from our long term development for a couple of months and explore just how much can be done by syntax alone. What is the general behavior of a purely syntactic parser and for what constructions is it strongest and weakest? This has given us a clearer idea of just where semantics and pragmatics would help the most.

We have chosen to implement a sentence-level parser which finds all parses of the sentence. We feel that, at least for the short run, this will be best for the practical question answering system we are trying to build. Our view is that current computer semantics and pragmatics will be too weak to determine in all cases the intended interpretation when they see it. For example, experience with the commercially offered ROBOT [14] system indicates problems such as confusion of the abbreviation *ME* for *Maine* with the pronoun *me* in sentences like (1). By finding both interpretations we can ask the user for a clarification.

(1) Print for me the sales of stair carpets.

When all parses are to be found it has generally been profitable to maintain a chart, or well-formed substring table. This approach has been carried fairly far in approaches based on standard

---

1 EQSP is the syntactic parser for an English query system currently under development

transformational grammar, but to our knowledge, no one has built a large parser for a grammar like LFG [19]. This we have done by adapting the well-known context-free algorithm of Earley to a very efficient form for parsing English.

Linguists have long viewed language as full of marked exceptions to powerful general rules. This we think is a general property of most large natural systems. No one algorithm can cover everything unless it is so general as to be terribly inefficient. The best approach to parsing is to split the problem into cases. There will be a primary algorithm (context free parsing in our case), important secondary algorithms (wh-movement and conjunction for us), and many minor algorithms (eg. for idiomatic expressions such as *per cent* and for arithmetic expressions). In this paper the reader will see how we have worked this out for the difficult MALHOTRA corpus.

As will be mentioned below, we have implemented our parser in a compiled rather than an interpreted form. This means the grammar is written directly as programs rather than as a data structure which is then interpreted by a separate parsing program. Richard Burton [4] first applied this technique, obtaining approximately an order of manitude improvement in parsing time over the interpreted implementation. We believe that we have relized a similar speed increase or perhaps a little more. One advantage of the compiled form is that it is easier to add special case processing algorithms. As stated above, we have added quite a few of these. It would be desirable to have a compiler program which could convert an abstract grammar into many different parser programs. This remains for further work.

There have always been questions of how complex natural language systems should be evaluated. One way, we think, is to run them on standard corpuses. This is why we give our results on the MALHOTRA and LADDER-TODS corpuses in the appendices. But in Section 4 we have tried to go beyond this and identify which constructions are the most difficult for our parser and why. A syntactic parser finding all parses always takes more resources per word as the length of the sentence grows. We have found specific sentence forms which demonstrate the best and worst cases, ie. they bracket the MALHOTRA data. For these sentences it is possible to analyze exactly why they are hard or easy for our parser. In this section we feel we have advanced, at least to some extent, the art of parser analysis.

It is very unlikely that people parse sentences the way our parser does. It proceeds breadth first keeping all options open. Nevertheless we feel a study of our paper may help people in cognitive science to appreciate the variety of issues that go into the construction of a parsing algorithm. In addition it is possible that people do proceed breadth first for one or a few words. One path may be considered most likely and brought to consciousness, but others could be under consideration in

parallel to facilitate local back-up. Anyone pursuing this idea may find our paper of some help. Finally, cognitive scientists may find our analysis of difficult constructions of some help in planning experiments, or they may want to run our parser to see just what syntactic structures certain sentences have in a LFG-like grammar. We are always surprised by some of the analyses found.

1. An Introduction to Chart Parsing

Let us define a very general context-free algorithm for comparison with historical landmarks such as Woods' ATN model [33] and Earley's algorithm [8, 9]. This discussion is intended as a review of the literature, in order to establish a common terminology which will be useful when we introduce some of our own ideas in the later sections. The organization of this section strongly follows that of [13, 27, 28].

A context-free parser takes as input a context-free grammar and a sentence and produces as output a chart[2] of labeled phrases. A labeled phrase is a sequence of words delimited by two brackets and labeled with a category symbol. Let the triple <i, j, c> denote a phrase of category *c* spanning the words between the i^{th} position and the j^{th}. For illustrative purposes these triples will be represented in a two dimensional matrix (i by j) as illustrated below for the sentence (2). For example, the entry {NP, VP} in Chart(2, 4) represents two analyses of the words between positions 2 and 4, namely [$_{NP}$ flying planes] and [$_{VP}$ flying planes]. (More efficient representations will be discussed shortly.)

(2) Input Sentence: $_0$ They $_1$ are $_2$ flying $_3$ planes$_4$

(3) Grammar:

N ⟶ they	V ⟶ are	N ⟶ flying	A ⟶ flying	V ⟶ flying	N ⟶ planes	
S ⟶ NP VP	VP ⟶ V NP	VP ⟶ V VP	NP ⟶ N	NP ⟶ AP NP	NP ⟶ VP	AP ⟶ A

(4) Chart:

0	{ }	{NP,N,they}	{S}	{S}	{S}
1	{ }	{ }	{VP,V,are}	{VP}	{VP}
2	{ }	{ }	{ }	{NP,VP,AP,N,V,A,flying}	{NP,VP}
3	{ }	{ }	{ }	{ }	{NP,N,planes}
4	{ }	{ }	{ }	{ }	{ }
	0	*1*	*2*	*3*	*4*

2 Some authors prefer the term *well-formed substring table* (wfst) to *chart*.

If there is a complete parse of the sentence, the chart will have an *S* in the top-most right hand corner to represent the fact that the parser found an S spanning the words from before the first one (0) to after the last one (*n*). Otherwise, the sentence is rejected. There are no entries in the lower left half of the chart because there no phrases which end before they start. The diagonal entries correspond to phrases of zero words (eg. trace and other empty categories).

There are many well-known parsing algorithms that produce a chart like that above in time $O(n^3)$ (proportional to the cube of the number of words). One such algorithm is given below. It finds a phrase between positions *i* and *j* by picking a position *k* in between and testing whether there are two phrases, one from *i* up to *k* and another from *k* to *j*, that can combine according to the grammar. For example in (2), the algorithm will determine that there is an S from 0 to 4 because there is an NP from 0 to 1 and there is a VP from 1 to 4 and the two phrases can combine according to the grammar rule S → NP VP. The general entry in the chart is:

(5) $$\text{chart}(i, j) = \bigcup_{i<k<j} \text{chart}(i, k) * \text{chart}(k, j)$$

where '$\alpha * \beta$' combines phrases from α and β according to the rules of the grammar. That is, it returns the set of phrases whose left daughter is from α and whose right daughter is from β. (For the present discussion we will assume that phrases have one or two daughters, or more formally, that the grammar is in Chomsky Normal Form [1].) This algorithm can be performed in $O(n^3)$ time by choosing all combinations of *i*, *j* and *k*, each of which have *n* possible values. (The multiplication step requires constant time, independent of the actual input words. It only depends on the grammar and hence it is often known as the *grammar constant*.)

```
for j := 1 to n do
    chart(j-1, j) := {A | A →* word_j}                    lexicon
    for i := j-2 downto 0 do
        chart(i,j) := ⋃_{i<k<j} chart(i,k) * chart(k,j)   invariant
if S is in chart(0,n) then accept
    else reject
```

This formulation of chart parsing is convenient for showing the parallelism between CF parsing and matrix multiplication. This is an important result, originally due to Valient [32], which allows us to take advantage of advances in matrix multiplication algorithms, currently a very active area of research in computer science. Intuitively, the parallelism comes from a very strong similarity in invariants; (6) is the invariant for chart parsing and (7) is the invariant for multiplying two matrices *A* and *B* to produce

a resulting matrix *C*.

$$(6) \quad chart(i,j) = \bigcup_k chart(i,k) * chart(k,j)$$

$$(7) \quad c_{ij} = \sum_k a_{ik} * b_{kj}$$

1.1 Enumeration Order

1.1.1 Depth First vs. Breadth First

This algorithm is similar to several well-known algorithms dating back to the early 1960's (eg. Harvard Predictive Analyzer [21] and the Cocke-Younger-Kasami [1]), though these algorithms tend to enumerate the chart in slightly different orders. From a computational point of view, the enumeration order is of surprising little importance. Sheil [29] showed that the $O(n^3)$ time bound can be achieved by any algorithm that limits its search space to points in $\langle i, j, k \rangle$-space. From this perspective, there is no difference between depth first enumeration of $\langle i, j, k \rangle$-space, breadth first enumeration, best first, or even random order, for that matter.[3]

So far, much of the psycholinguistic literature has concentrated on basically depth first (serial) models [17, 10, 23, 6] for reasons that are intuitively appealing, though difficult to formalize. The parser to be presented here is an extreme alternative to depth first; it is completely breadth first. Though we don't believe this particular enumeration order to be realistic with respect to psycholinguistics, it offers a useful milestone for comparison with the more popular depth first models. If one wanted to argue that depth first was realistic on functional grounds, then he ought to be able to show that it is computationally more efficient than breadth-first. We find plausible the hypothesis that human processing proceeds breadth-first for a few words before selecting the best alternative to follow in a depth-first fashion; our parser may be helpful to someone exploring such a compromise position.

---

3 This claim needs a slight qualification; it doesn't hold if the equality relation in the invariant is replaced with an assignment statement because the former is associative while the latter is not. That is, every order of evaluating equality relations produces the same results, but this is not necessary true of assignment statements. Hence the equality sign in the algorithm really means equality and not assignment. (There is a recent trend in computer science to replace assignment statements with equality constraints [31], thus side-stepping a large number of ordering problems and similarly, in linguistics there is a trend to replace strict ordering of transformations with well-formedness constraints such as Chomsky's conditions on binding, case, government, and thematic relations [5] or Bresnan Kaplan's completeness, coherence, and consistency [19].

We have chosen an extreme breadth-first position for two reasons. First, it happens to be very convenient for certain representation issues. Secondly, because we want to find all parses of multiply ambiguous sentences, there are few (if any) advantages of depth-first. Depth-first has better average case behavior if only one parse is desired, because it can stop searching when it finds the first parse, but this doesn't apply in the "all-parses" problem. When seeking all parses, it is more productive to discuss the search space itself, rather than particular strategies of enumerating the space.[4]

1.1.2 Top-down vs. Bottom-up

The psycholinguistic literature has also paid considerable attention toward the top-down/bottom-up question. This distinction becomes considerably less important when a chart is introduced, because each phrase will be found once and only once, for both enumeration strategies. Without a chart, the top-down/bottom-up distinction might be much more important because a strictly top-down parser will find a phrase multiple times, once for each place that it can be used in a larger phrase whereas a strictly bottom-up parser will find a phrase just once. However a strictly bottom-up parser has the dual problem of proposing a larger phrase multiple times, once for each way that it can use a phrase that has been found. All efficient parsers combine both top-down and bottom-up information in some way (using the chart, for example), so that it is useless to classify these parsers as one type or the other. This may be part of the reason that this question is so difficult for psycholinguistics to resolve. This paper will concentrate more on representation issues where it is much easier to formulate sharp distinctions.

For example, is the chart algorithm above top-down or bottom-up? Well, most people would say that it is bottom-up because it finds phrases from i to k and from k to j before it puts them together to form a phrase from i to j. However, it is fairly easy to reformulate the multiplication step so that it looks more like division. That is, one might replace a statement like (8) with something more like (9). (From a formal grammar point of view, these pairs contrast Chomsky Normal Form [1] with Categorial Grammars [2], and from a representational point of view, these pairs suggest two different ways of indexing[5] the grammar.)

---

4. It may be useful to think about the search space as the *competence* of processing and the search strategies as the *performance* of processing.

5. *Indexing* is a technical term borrowed from data-base management. Data-bases are indexed on certain keys so that it is relatively easy to find an item if you have its key. A data base index is analogous to an index at the back of a book, both make it relatively easy to find a particular item if you know what it might be indexed under.

(8) $chart(i, j) = chart(i, k) * chart(k, j)$ *bottom-up (Chomsky Normal Form)*

(9) $chart(k, j) = \frac{chart(i, j)}{chart(i, k)}$ *top-down (Categorial Grammars)*

Earley's Algorithm [8] can be viewed as (9) because it predicts phrases from the top (chart(i, j)) and then divides (completes) them by phrases from the bottom (chart(i, k)) so that it can predict the next phrase (chart(k, j)). Since the predict step precedes the complete step, this algorithm is often said to be top-down, though it is really very similar to the chart parser presented here, and hence one really wouldn't want to say that it has a completely different search strategy. Top-down/bottom-up arguments are unconvincing because (8) and (9) are so similar to each other that it seems extremely tenuous at best for an argument to depend crucially on a distinguishing trait. In fact they are so close that it is often possible to make one strategy look like another by a clever recoding trick; one will be provided for Earley's Algorithm at a later point. As noted previously, it is more productive to discuss the search space itself, rather than enumeration orders.[6] Our analysis will attempt to follow this approach in general, though we will mention a particular search strategy (bottom-up and breadth first) in the representation discussion for concreteness. The reader is invited to reformulate those results in terms of a top-down depth first strategy.

1.2 N-ary Branching

The previous algorithm has a rather awkward assumption that all phrases have only one or two daughters (Chomsky Normal Form). This would appear to conflict with many linguistic analyses which propose wider branching factors in cases such as:

(10) $[_{vp}\ [_{v}$ give$]\ [_{np}$ it$]\ [_{pp}$ to him$]]$

This section will relax this assumption by recoding a general context-free grammar into Chomsky Normal Form in such a way that it should be clear that there are no linguistic and/or psycholinguistic implications resulting from this assumption. In so doing, we will have generalized the previous algorithm to work for all context-free grammars.

---

6. Much of the emphasis on enumeration orders stems from the heuristic search paradigm which is traditionally used in Artificial Intelligence to improve performance by imposing certain cut-offs in the enumeration procedures, thus finding an approximate solution in less time. We would rather view the heuristics (when possible) as shrinking the search space itself, enabling a *complete* enumeration of the reduced space. Thus we can deal with heuristics without bringing up processing notions like enumeration orders.

The recoding trick is very simple. Define a *dotted-rule* to be a context-free grammar rule with a dot inserted to indicate how much of it has been parsed. For example in (11)-(14) below, the dotted-rules on the left correspond to the tree fragments on the right.

	dotted rule	tree fragment
(11)	VP →. V NP PP	$[_{vp}$
(12)	VP → V. NP PP	$[_{vp}$ give
(13)	VP → V NP. PP	$[_{vp}$ give $[_{np}$ it]
(14)	VP → V NP PP.	$[_{vp}$ give $[_{np}$ it] $[_{pp}$ to him]]

Since this notation is fairly cumbersome, it will be convenient to introduce a few abbreviations. When there can be no confusion, we will abbreviate initial and final dotted-rules ((11) and (14) respectively) by placing the dot on the category itself (eg. .VP and VP.) rather than spelling out the entire rule. Other abbreviations will be introduced as they become useful.

The multiplication rule will be redefined to combine dotted-rules rather than nonterminals as before. It combines two dotted-rules, a partial and a final, by moving the dot past the next symbol as illustrated below:[7]

(15) {VP →. V NP PP} * {V.} = {VP → V. NP PP}
{VP → V. NP PP} * {NP.} = {VP → V NP. PP}
{VP → V NP. PP} * {PP.} = {VP → V NP PP.}

It might be useful to view these dotted-rules as forming the nonterminal of a new grammar which we will call the *dotted-grammar*. The combination rule above is constructed to have binary branching so the dotted-grammar will be in Chomsky Normal Form. Now the algorithm above can be applied to the dotted-grammar instead of the original, and thus meeting the binary branching assumption without imposing any constraints on the original grammar. The resulting algorithm is a general parser for any context-free based theory of processing since the dotted-grammar is a trivial mapping from the original which imposes no linguistic and/or psycholinguistic restrictions.

---

7. Technically the multiplication rule is defined to apply to *sets* of nonterminals, rather than individual nonterminals, and consequently, dotted rule multiplication really applies to sets of dotted rules, not individual rules. This explains the set brackets in the example

1.3 Dotted-grammars and ATN States

One particularly well-known framework is Woods' Augmented Transition Network (ATN) [33] and hence it is worthwhile to reformulate the preceding explicitly in his terms. He introduces his model in three stages. First he begins with finite state transition networks which consist of a set of states connected by labeled arcs. The interpreter starts from the initial state and follows arcs labeled with the category of the first input word. From there it follows arcs labeled with the category of the second input word and so on until there are no more input words. If the machine is in the final state when there are no more words, the sentence is accepted, and if not, the sentence is rejected. This machine is strongly equivalent to a regular grammar.

Woods then increases the generative capacity by allowing arcs to specify other networks recursively. For example, the S network could specify an NP network for its first transition, not just a lexical category as in the finite state network model. Woods called this recursive model a *Recursive Transition Network* (RTN) and showed it to be strongly equivalent to context-free grammars. He then augmented it with registers and conditional jumps to construct the *Augmented Transition Network* (ATN) which has the power of a Turing Machine (though there have been efforts to reduce its generative capacity).[8]

There is a strong relationship between Woods' RTNs and dotted grammars. RTN states correspond to sets of dotted-rules, and arcs perform the multiplication as illustrated below. (This paper will use the terms *state* and *dotted-rule* interchangeably.)

states	arcs	*RTN*
{.VP}	push {V.} to {VP → V. NP PP}	
{VP → V. NP PP}	push {NP.} to {VP → V NP. PP}	
{VP → V NP. PP}	push {PP.} to {VP.}	

dotted rules	multiplication table	*Dotted Grammar*
{.VP}	{.VP} * {V.} = {VP → V. NP PP}	
{VP → V. NP PP}	{VP → V. NP PP} * {NP.} = {VP → V NP. PP}	
{VP → V NP. PP}	{VP → V NP. PP} * {PP.} = {VP.}	

The ATN employed a convenient naming convention for partial constituents (intermediate states) that will be used here when there can be no confusion. Let S/NP abbreviate an S up to the NP (S → NP. VP) and VP/V abbreviate a VP up to the V (VP → V. NP, VP → V. AP or VP → V. NP). This abbreviation scheme

8. [Kaplan (personal communication)]

combines several dotted-rules that are identical up to the dot. This is often a useful optimization, though there may be times when it is more efficient to distinguish them. For example, it might be better to combine dotted-rules that are the same *after* the dot.

1.4 Example I (with Dotted-Rules)

The chart with dotted-rules is given below for the sentence: *They are flying planes.*

0	.S, .NP, .VP, .AP .N, .V, .A	NP., N., S/NP they	S.	S.	S.
1		.S, .NP, .VP, .AP .N, .V, .A	V., VP/V are	VP.	VP.
2			.S, .NP, .VP, .AP .N, .V, .A	A., AP., NP/AP, V., VP/V N., NP., S/NP, flying	NP., S/NP, VP.
3				.S, .NP, .VP, .AP .N, .V, .A	NP., N. S/NP planes
4					.S, .NP, .VP, .AP .N, .V, .A
	0	*1*	*2*	*3*	*4*

The diagonal of the chart takes on a very important meaning. Before it was only used for linguistically empty phrases (e.g. traces). Now, it is also used for initial dotted-rules (rules with the dot in the left most position) because they denote partial phrases dominating no words in the input stream ($i = j$). By similar reasoning, there will be no initial dotted-rules in the off-diagonal entries. Traces are now denoted by final dotted-rules in a diagonal entry.

2. Taking Advantage of Restricted Grammars

Though the $O(n^3)$ bound is approximately the best known for the general context-free parsing problem,[9] it is possible to do much better with certain restricted grammars. For example, Earley's Algorithm [8, 9] parses some context-free grammars in O(n) time, some others in $O(n^2)$ time and the rest in $O(n^3)$ time. This is an improvement over the algorithm outlined above which cannot take advantage of these restricted grammars; it requires $O(n^3)$ time in all cases because it enumerates $O(n^3)$ points in $\langle i, j, k \rangle$-space even if many of them cannot enter into the complete parse due to some restriction in the grammar. Pratt [26, 27] and Ruzzo [13, 28] have independently found a way to incorporate Earley's ideas into an algorithm like the one we first presented; the modified algorithm will

---

9 Valient [32] found an O(M(n)) bound where M(n) is the time required to multiply two matrices of size *n*. This is known to be slightly better than cubic, though the constants are probably prohibitive for practical applications.

use the grammar to decide which points to enumerate and hence it can be much more efficient for certain restricted grammars. The next two sections will illustrate two ways that the grammar can restrict the set of reachable points in <i, j, k>-space.

2.1 Time n^2 Grammars

In general, there are O(*n*) ways to pick *k* so that it partitions the words between *i* and *j* into two phrases that can combine to form a single phrase from *i* to *j*. However, for some grammars there are only a bounded number of ways of picking *k*; the grammar will not combine the other partitions and hence there is no reason to consider them. This is a particularly useful restriction because if there are only *b* ways to pick *k*, then there are only O(b * n * n) points in the <i, j, k> search space. Earley [8] called this restriction *Bounded Direct Ambiguity*, observing that these grammars have limited "top-level" ambiguity. (They can still be very ambiguous because each of the *b* top-level partitions can themselves have *b* partitions, and each them can have *b* partitions, and so on.)

We will give an example and then compare it against the more general case. Then we will draw on some examples from the psycholinguistic literature and speculate that they might be interesting subcases of this restriction. Finally it will be shown how the algorithm can be modified so as to take advantage of this restriction. The modified grammar takes $O(n^2)$ time on restricted grammars, though it still requires $O(n^3)$ in the general case.

2.1.1 Grammar 'aAa': An Example of Bounded Direct Ambiguity

Consider grammar (16) which parses odd length strings of *a*'s. This grammar is an extreme case of bounded direct ambiguity; there is only one way to pick *k* between *i* and *j* because the grammar is unambiguous. There are less extreme cases which are ambiguous, though this is a good example to begin with.[10]

(16) A → a A a
A → a

---

10 In fact, any *linear grammar* has bounded direct ambiguity. (These grammars have at most one nonterminal on the right hand side of a production.) See [8] for the proof.

The following abbreviations will be used for this example:

abbreviation	long forms
.A	{A → .a, A → .a A a}
A/a	{A → a., A → a. A a, a}
A/A	{A → a A. a}
A.	{A → a A a.}

The chart is given below for the sentence: "aaaaa".

0	.A	A/a	A/A	A.	A/A	A.
1		.A	A/a	A/A	A.	A/A
2			.A	A/a	A/A	A.
3				.A	A/a	A/A
4					.A	A/a
5						.A
	0	*1*	*2*	*3*	*4*	*5*

Notice that there is only one way (at most) to pick k so that it partitions a set of input words into two phrases that can combine. For example, between 0 and 5, only 4 will do (A/A * A/a = A.); for any other k, chart(0, k) * chart(k, 5) = { }. This restricted grammar is parsed more efficiently by Earley's algorithm because there are fewer k's to look for; it avoids looking for combinations that can't exist. Notice that this improvement saves time, but doesn't save any space; the improved algorithm will find just as many phrases as the original algorithm did. (There is a common misconception that Earley's algorithm is faster just because it avoids constructing unnecessary phrases. This explanation is incomplete because it doesn't account for the improved performance in cases like this where there are still $\Omega(n^2)$ phrases,[11] just as many as there were before.)

2.2 Grammar 'AA': An Example of Unbounded Direct Ambiguity

Contrast the previous example with one where there is no bound on the direct ambiguity. With a grammar like (17) below, all choices of k will work out and hence Earley's algorithm will have to look at all of them. This is one of the worst grammars for Earley's algorithm; it requires n^3 time.

---

11. See [20] for a formal definition of this notation; it means that the growth is at least as fast as n^2. This is a lower bound; before we were discussing upper bounds when we said that the growth was no faster than n^2.

(17) $A \rightarrow A\ A$
$A \rightarrow a$

The following abbreviations will be used for this example:

abbreviation	long forms
.A	{A → .A A, A → .a}
A/a	{a, A → a., A → A. A}
A.	{A → A. A, A → A A.}

The chart is given below for the sentence: "aaaaa"

0	.A	A/a	A.	A.	A.	A.
1		.A	A/a	A.	A.	A.
2			.A	A/a	A.	A.
3				.A	A/a	A.
4					.A	A/a
5						.A
	0	*1*	*2*	*3*	*4*	*5*

2.3 Examples from Psycholinguistic Literature

Psycholinguists have looked at quite a number of English constructs which have unbounded direct ambiguity because they seem to provide some interesting differences between performance and competence. It appears that unbounded direct ambiguity is difficult to process, a result that is compatible with the theoretical discussion above. Furthermore, many computational linguists have also found these constructs problematic for their respective models. These constructs are especially difficult for Marcus' Determinism Hypothesis [23]; he and many of his followers [23, 6, 25, 30] have worked out possible solutions to many of them, usually concluding that they have bounded direct ambiguity in performance.[12] Some experience with our own model will be presented.

---

12 There is one exception; Church [6] has experimented with an alternative approach called *pseudo-attachment* which attaches a phrase to an unbounded number of places in a single step. In certain cases, this approach has been encoded directly into the grammar of the parser to be presented here

2.3.1 Noun-Noun Modification

This case is presented first because it is perhaps the closest analog of 'Grammar AA', though it may not be the most important from a *practical engineering point of view*. These examples tend to involve domain specific semantics; perhaps they are best resolved at some level other than syntax.[13] (These particular examples were taken from [23].)

(18) NP → NP NP *grammar*
NP → N

(19) [[[water meter] cover] adjustment] screw]

(20) [[ion thruster] [performance calibration]]

(21) [[boron epoxy] [[rocket motor] chambers]]

(22) [1970 [[balloon flight] [[solar-cell standardization] program]]]

These are a well-known problem for processing. Some of the difficulties are discussed in [23] and references therein; Robert Milne (personal communication) is currently working on some solutions within a deterministic framework. This construction is also problematic for nondeterministic systems because it requires considerable resources. The approach taken here is to flatten the syntactic structure of these phrases as in (23), delaying the decisions for semantics. In this way, the parser will not waste time trying all possible bracketings; it will be content with a canonical one that represents them all. This is very similar to the "pseudo-attachment" approach in [6]. Notice that the canonical grammar has bounded direct ambiguity.

(23) NP → (N)* *canonical grammar*

2.3.2 Prepositional Phrase Attachment and Conjunction

There are many other cases of 'Grammar AA' from natural language; perhaps the most common are prepositional phrase attachment and conjunction.

---

13. We haven't decided yet how semantic processing should be ordered with respect to syntactic processing. Although the current EQSP system orders all syntactic processing first, we plan to experiment with some more interactive scheduling strategies

(24) NP → NP and NP — *Grammar for Conjunction*
NP → DET N

(25) NP → NP PP — *Grammar for Prepositional Phrase Attachment*
NP → DET N
PP → P NP

Both of these grammars have unbounded direct ambiguity; there are an unbounded number of choices for *k*. We have enumerated the top-level choices of *k* for the following two examples. Notice that the number of choices grows with the number of words. (Let NP/and and NP/NP abbreviate the dotted-rules: NP → NP and. NP and NP → NP. PP, respectively.)

(26) plant 1 and $_1$ plant 2 and $_2$ plant 3 and$_3$ plant 4 and $_4$ plant 5 — *conjunction*
[$_{NP/and}$ plant 1 and] [$_{NP.}$ plant 2 and plant 3 and plant 4 and plant 5]
[$_{NP/and}$ plant 1 and plant 2 and] [$_{NP.}$ plant 3 and plant 4 and plant 5]
[$_{NP/and}$ plant 1 and plant 2 and plant 3 and] [$_{NP.}$ plant 4 and plant 5]
[$_{NP/and}$ plant 1 and plant 2 and plant 3 and plant 4 and] [$_{NP.}$ plant 5]

(27) plant 1 $_1$ with plant 2 $_2$ with plant 3 $_3$ with plant 4 $_4$ with plant 5 — *PP attachment*
[$_{NP/PP}$ plant 1] [$_{PP.}$ with plant 2 with plant 3 with plant 4 with plant 5]
[$_{NP/PP}$ plant 1 with plant 2] [$_{PP.}$ with plant 3 with plant 4 with plant 5]
[$_{NP/PP}$ plant 1 with plant 2 with plant 3] [$_{PP.}$ with plant 4 with plant 5]
[$_{NP/PP}$ plant 1 with plant 2 with plant 3 with plant 4] [$_{PP.}$ with plant 5]

In both cases, the number of parses grows very quickly with the number of phrases. The actual growth is exactly the *Catalan numbers* [20], the number of ways to insert parentheses into a formula of *n* terms. The first few Catalan numbers are: 1, 2, 5, 14, 42, 132, 469, 1430, 4862, ... They are generated by $\binom{2n}{n} - \binom{2n}{n-1}$ which grows almost exponentially. These numbers have been empirically verified with up to nine prepositional phrases. That is, the parser found 4862 parses for the sentence: *It is the number of products of products of products of products of products of products of products of products of products.* The parser uses a much more complicated grammar for conjunction; consequently, it is harder to predict the number of parses in that case.

The canonical grammar approach (proposed in the last subsection) is also applicable here; this is similar to the "pseudo-attachment" approach taken in [6]. This approach was not implemented for prepositional phrases in this work because there are rarely more than three of them in a sentence, which is only five ways ambiguous, and hence it does not seem to be worth the effort to canonicalize them. The canonical grammar approach was implemented in some cases of conjunction; it appears to save considerable amount of work in this case because sentences often contain more than three potential conjuncts.

2.3.3 Reduced Relative Clauses

It appears that one of the greatest proliferations of ambiguity is due to reduced relative clauses. Not only do they tend to attach in every possible way (like prepositional phrases and 'Grammar AA'), but they can also start wh-movement and they are often confused with main verbs. Furthermore, if that isn't bad enough, a reduced relative is often just one word long and hence it is one of largest contributors to the *per word ambiguity* (number of parse trees divided by sentence length). For example, sentence (28) is 10 ways ambiguous, but without the word *produced*, it is only 3 ways ambiguous. The effect is far more significant in (30)-(31) where there are two of them to interact with a few more phrases. These sentences do not appear to be very difficult for most people to understand. Perhaps people are using semantic constraints to reduce the direct ambiguity.

(28) List the sales of products *produced* in 1973. *10*

(29) List the sales of products in 1973. *3*

(30) List the sales of products *produced* in 1973 with the products *produced* in 1972. *455*

(31) List the sales of products in 1973 with the products in 1972. *28*

Reduced relative clauses don't appear to have the same direct ambiguity in performance as they do in competence, especially when they can be interpreted as main verb phrases, as well-noted in the psycholinguistic literature by a pair like the following:

(32) # The horse raced past the barn fell.
The horse ridden past the barn fell.

Many researchers have argued that the unacceptability of sentence (32) is due to some difficulty in locating the boundary between the noun phrase and the verb phrase. In our terms, we would say that it is difficult to find the k which partitions the sentence into a noun phrase and a verb phrase. Notice that the general case (33) has unbounded direct ambiguity.

(33) The horse raced past the horse raced past the horse ... fell.

The general problem with all of these constructions (reduced relatives, prepositional phrases, conjunction, noun-noun modification) is unbounded direct ambiguity and hence there is a characteristic proliferation of interpretations (Catalan growth) and significant time requirements (cubic growth). The general solution to this problem is to find other constraints (semantic, performance, canonicalization) in order to bound the direct ambiguity. The parser to be presented

here is the result of considerable efforts in these directions; it generates far fewer parse trees than it once did. It becomes particularly clear just how many parses there are when one is generating all parses, rather than just the first one as most other parsers do.

2.4 Taking Advantage of Bounded Direct Ambiguity

This section will modify the above algorithm to take advantage of bounded direct ambiguity. The improvement results from ignoring null entries in the chart. Instead of enumerating all points in <i, j, k>-space, the improved algorithm will only enumerate those *k*'s that can partition the words from *i* to *j* into two phrases that can combine. It accomplishes this by working "bottom-up";[14] that is, it will combine two phrases only if they are both already in the chart. Whenever the parser adds a new phrase into the chart from *k* to *j*, it looks for phrases ending at *k* and combines them to form phrases from *i* to *j*. These phrases are then completed with phrases ending at *i*, and so on. The combination procedure is called *completer* (Earley's terminology) because it completes phrases already in the chart. (For technical reasons, the chart is initialized with zero length phrases already inserted along the diagonal because they can be precomputed without looking at the input and because the completer, as defined here, is unable to find them. This technical point will be cleaned up shortly. Recall that initial states are zero length phrases and consequently, they are all initially in every diagonal entry.)

We can make the completer somewhat faster by precomputing which categories can combine with which. Then the completer will only try to combine phrases of the "right" categories; that is, instead of looking for just any phrase ending at *k*, it looks for phrases of a category that can combine with the category of the phrase from *k* to *j*. The function *left_sisters_of* returns a precompiled list of dotted-rules that can combine with a given dotted-rule. So for example, it would return .S and VP → V. NP PP and a host of others as left sisters of NP., all of which multiply on the left of an NP. to produce some other states.

This algorithm motivates a representation that allows efficient enumeration of phrases of a particular category ending at a particular point. These representation issues will be discussed later. For now, we will assume a few functions for manipulating the chart. These functions now take three arguments (a *state*, an *i* and a *j*) unlike the previous chart functions which did not take a *state* argument. The function *chart(s,?k,j)* generates a list of *k*s, one for each phrase of category *s* ending at *j*.

---

14 Note that this algorithm can also be viewed as top down for reasons mentioned previously, top down information is used to decide what should be put into the chart in the first place.

```
(34) parse: proc()
         for j from 1 to n
             for s in {A. | A →• word_j} do
                 add_chart(s,j-1,j)

     add_chart: proc(s,i,j)
         if chart(s,i,j) = 'not found' then
             chart(s,i,j) := 'found'
             complete(s,i,j)

     complete: proc(s_2,k,j)
         for s_1 in left_sisters_of(s_2) do
             for i in chart(s_1,?i,k) do
                 add_chart(s_1*s_2,i,j)
```

2.5 Time n Grammars

It has been noted [8, 27, 28] that the preceding algorithm is inefficient for certain restricted grammars because it constructs phrases that cannot be used in a complete parse. For example, given the grammar 'Aa' (below), it will construct approximately n^2 phrases, most of which do not fit into a final parse. The chart is given below with underlining used to denote useless phrases. These phrases do not fit into a parse that spans from 0 to *j* and hence they are *useless*. (This *usefulness constraint* is enforced by the *predict* operation in Earley's algorithm.) Notice that there is only a linear number of useful phrases; there are *n* useful phrases on the top row and there are *n – 1* on the off-diagonal. This is typical of left branching grammars because the top row contains phrases starting at the left edge and the off-diagonal contains single word phrases. Consequently, the chart can be represented in linear space, which is a large improvement in space complexity over the general n^2 bound. The time bounds are also reduced to O(n) as we will see.

(35) $A \rightarrow A\ a$ *Grammar Aa*

$A \rightarrow a$

abbreviations	long forms
.A	{A → .A a, A → .a}
A/A	{A → A. a}
a.	{A → a., A → A. a}
A.	{A → A a.}
A.*	A. ∪ A/A

0	.A	a.	A.*	A.*	A.*	A.*
1		.A	a.	A.*	A.*	A.*
2			.A	a.	A.*	A.*
3				.A	a.	A.*
4					.A	.a
5						.A
	0	*1*	*2*	*3*	*4*	*5*

The usefulness condition reduces the complexity for a large class of grammars including all deterministic grammars[15] with one extra modification called *lookahead*. The usefulness condition requires a phrase to be consistent with everything from position 0 to *j*. The lookahead modification strengthens this to include a few more tokens to the right of *j*. This condition is important for right branching grammars which require n time with lookahead, but n^2 time without (on Earley's algorithm). For example, consider a sentences such as (36) which has a right branching grammar similar in structure to (37).

(36) He believes he believes he believes it.

(37) $A \rightarrow a\ A \mid a$

This sentence can be parsed in linear time on Earley's algorithm if the multiplication step looks ahead one token, but otherwise it is just like Grammar 'aAa' which consumed square time. Lookahead is crucially used in this grammar to determine the product of {.A } * {a}. If there is another *a*, then the result is {A → a. A}, but if not, the result is {A.}. Without lookahead, the result would be {A → a. A, A.} in both cases and consequently the algorithm would not be taking advantage of the fact that a completed *A* cannot be followed by another *a* in the input string. In these right branching

---

15 Formally, a deterministic grammar is a grammar that can be recognized on a deterministic push down automaton (DPDA). This is slightly different from Marcus' notion whose machine is more powerful. Deterministic grammars can generate all finite state languages and almost all computer languages (eg LISP). However, they are unambiguous and hence they are not a good candidate for natural language and certain older computer languages (eg FORTRAN).

grammars, the lookahead condition has the same effect that usefulness has in left branching grammars; that is, it reduces space bounds from n^2 to n by making the parser behave deterministically. This also improves the time bounds by a factor of n.

This lookahead condition has not been used in our parser except in a few cases such as arithmetic expressions, numbers, ordinals, hyphenated words, and certain conjunction phrases (eg. *and also*). We believe there are very few constructions in our grammar where lookahead helps. It is crucially important for deterministic parsing of right branching structures [23, 6], though most of our right branching structures are ambiguous and hence it doesn't do much good to look ahead. It really only helps to look ahead if the parser is going to behave differently based on what it finds. But since our parser is going to find all paths anyway, it doesn't do much good to look ahead and find out that the structure is ambiguous. However, there are a few unambiguous right branching constructions such as (36) where EQSP should look ahead. We will show in section 4 that EQSP pays the price for not looking ahead; that is, it takes another factor *n* longer to parse sentences like (36) than it would if it looked ahead. Nevertheless, even without lookahead, the class of time *n* grammars is very large. Earley [8] showed the class to properly include LR(0) grammars (deterministic grammars that don't need lookahead).

2.5.1 Taking Advantage of Useless Phrases

Earley's parser avoids looking at useless phrases by excluding them from the chart and thus restricting the parser's attention to just useful phrases and no others. There are three cases of a useful phrase:

(38) At position 0, .S is useful. *initial condition*
(39) It is the combination of two other useful phrases.
(40) It is the initial state of a phrase that can combine with a useful phrase. *Earley's predictor*

Earley calls this third condition, *the predictor*, though this term is somewhat misleading. For example this third condition will cause .NP to become useful if .S were useful because .S can combine with NP. and .NP is the initial state of an NP. This will then cause .N to become useful because .NP can combine with an N. Notice that the new states (eg. .NP and .N) are both the initial state of a right sister of an already useful state. For example, .NP is the initial state of NP. which is a right sister of .S. (A right sister combines on the right by analogy to a left sister which combines on the left.)

These "predictions" fall on the diagonal of the chart because they are initial states; the other useful phrases fall in the upper triangle. Previously the chart was initialized with the diagonal containing all

initial states; the modified algorithm below will add these initial states when they are known to be useful, cleaning up the previous version of the algorithm.

```
(41) parse: proc()
         add_chart({.S}, 0, 0)                                  Initially .S is useful
         for j from 1 to n
             for s in {A. | A →• word_j} do
                 add_chart(s,j-1,j)

     add_chart(s,i,j)
         if chart(s,i,j) = 'not found' then
             chart(s,i,j) := 'found'
             for prediction in right_sisters_of(s) do                Earley's Predict
                 add_chart(initial_state_of(prediction),j,j)
             complete(s,i,j)

     complete: proc(s2,k,j)
         for s1 in left_sisters_of(s2) do
             for i in chart(s1,?i,k) do
                 add_chart(s1*s2,i,j)                    combination of useful phrases
```

This new algorithm takes linear time for left branching grammars because there is only a constant number of phrases for each *j*, and each of them can combine with only a constant number of phrases (*chart*(s_1,?*i*,*k*) is bounded by assumption) and hence the total time is constant for each word, which is linear with the sentence length. Similar arguments apply for all LR(0) grammars. The algorithm could be extended to parse LR(k) grammars (deterministic grammars with lookahead of *k*)[16] in linear time by adding lookahead.

2.6 Representation Issues

The representation of the chart becomes very important for efficient parsing. There are only two operations that reference the chart in the above algorithm:

```
(42) for i in chart(s,?i,j) do                                      enumeration

(43) if chart(s,i,j) = 'not found' then                                 check
         chart(s,i,j) := 'found'
```

16. The *k* lookahead is different than the *k* which partitions *i* and *j* into two phrases.

The analysis requires step (42) to take time proportional with the number of phrases found and step (43) to take constant time. This could be assured if the chart allowed random access on *s* and *j*. For example, the chart could be a two dimensional array, the first dimension ranges over choices of *s* and the second dimension over choices of *j*. Each entry in the array would be a set of *i*'s.[17] Unfortunately, this array is probably too large for long sentences and hence it is often *hashed* in practice.[18] We have implemented an alternative proposal to the space problem; suppose the chart does not allow random access on *j*, but rather, the chart allows random access on just *s* and then sequential access on *j* so that the most recent *j*'s are available first. That is, for each *s* there is a list of <i, j>-pairs, sorted on *j*. An example is worked out below (except for diagonal entries terminals (words) which are represented differently).

(44) $_0$ They $_1$ are $_2$ flying $_3$ planes. $_4$

rule	edges	rule	edges
S.	(<0, 4>, <0, 3>, <0, 2>)	V.	(<2, 3>, <1, 2>)
VP.	(<1, 4>, <2, 4>, <1, 3>)	A.	(<2, 3>)
NP.	(<2, 4>, <3, 4>, <0, 1>)	S/NP	(<0, 1>)
AP.	(<2, 3>)	VP/V	(<2, 3>, <1, 2>)
N.	(<3, 4>, <0, 1>)	NP/AP	(<2, 3>)

Now, in order to enumerate phrases of category *s* ending at *j*, the parser fetches the list of phrases of category *s* and then searches the list for those ending at *j*. The search halts when a phrase on the list ends before the desired *j*. This representation is taking advantage of the fact that a breadth first search is almost always referencing the more recent phrases, and only very rarely will it complete a phrase ending very far back. This representation would be very inefficient for a depth first parser (if it has to back up very often) because the chart would contain a large number of phrases ending after the desired *j*.

The above example does not mention terminals (words) or initial dotted-rules. It isn't necessary to represent terminals (words) in the chart explicitly if they are always dominated by a nonterminal (eg. N, V, A) as they are in our grammar. The next subsection will discuss initial dotted-rules.

---

17 This doesn't actually allow constant time *check* because the list of *i*'s might become very long, but it is fairly close since the most interesting values are kept near the front of the list in practice. If one were concerned with the theoretical behavior, he could also add a second table for checking which would be accessed differently from the table for enumeration.

18 *Hashing* is a popular technique of compressing very sparse arrays. See [20] for an excellent analysis of its theoretical properties.

2.6.1 Diagonal Entries

Diagonal entries are represented differently for four reasons:

(45) They tend to be much denser
(46) They are checked more often
(47) They are not enumerated
(48) They contain less information

Notice that there can be at most one diagonal phrase for each *s* and *j* (because *i* equals *j* on the diagonal), and hence, only one bit is needed to represent whether or not that phrase is in the chart. Consequently these entries are stored in a bit array that allows random access on both *s* and *j*. These bit arrays are a commonly used technique for representing initial predictions in many algorithms (eg. [27, 28, 13]). There are two exceptions where we represent a diagonal entry with the standard data structure, not the bit arrays. The first exception is when a diagonal entry is being conjoined and the second is when it is starting a relative clause (for wh-movement). These cases will be explained in more detail when we discuss conjunction and wh-movement. There are a few other cases where we don't bother to store the diagonal entries because these constructions are almost always useful, and hence the parser simply assumes that they are useful without checking the chart.

construction	example
classifier-name-phrase	They are dept 2 red dresses.
conjunct-phrase	even though; and also
head-name-phrase	W A MARTIN
ie-phrase	ie dogs, ie in the woods
other-phrase	other than red; other than me
ordinal	3rd; 4th
very-phrase	very quickly
per cent	*per cent*
hyphenated-word	fire-plug

Actually the representation of edges is slightly more complicated; they are really records (random access) including several other fields in addition to *i* and *j*. In particular, there is a field for features (eg. person, number, etc.) and a field for a wh-element (the hold cell), and there is a slot for conjunction. These fields will be discussed in more detail in a later section.

2.7 Compilation vs. Interpretation

All of the algorithms presented so far *interpret* the grammar more or less directly as is, with almost no pre-processing. In fact, only the left-sisters and/or right-sisters have to be computed in advance; the rest of the grammar can be interpreted directly in context-free rule format. On the other hand, one could imagine a parser which pre-*compiled* the grammar into some other format which was more efficient for processing. The compilation process is a one-time operation which precedes the parsing of any sentences. This approach can provide considerable efficiency improvements as Burton has shown in his thesis [4].

Compilation is a very common notion in computer science where it is used to transform high-level programs into more efficient machine-level programs before they are run on any input. Compilation is to be contrasted with interpretation where there is no pre-processing; all transformations are performed as they become necessary depending on the input data. Compilation avoids performing the same transformations again and again; interpretation avoids preparing for input data that will never occur.

We have compiled the grammar[19] so that there is an *add_chart* procedure for each state in the grammar. This makes it possible to pre-compile the loop (49) into a single bit vector operation which marks a number of states useful in parallel. There are a number of other advantages from this move. For example, there is no need to try to complete a state unless it is the final state of some network. Therefore, the calls to the complete procedure are removed from the add_chart procedures except for the add_chart procedures of final states.

```
(49)  for prediction in right_sister_of(s) do
          add_chart(initial_state_of(prediction),j,j)
```

We have found these very simple compilation techniques to be extremely important in improving the runtime efficiency of the parser in exchange for a large increase in program size.

---

19. Ideally the compilation would be performed by a program so that it would be relatively easy to modify the grammar. In this preliminary project, we have compiled the grammar by hand in order to avoid the very difficult task of designing a grammar compiler.

3. Transformations and Lexical Rules

So far we have limited our attention to pure RTNs; this chapter will discuss a number of useful augmentations which capture a wide number of linguistic generalizations. Almost all of these facts could be captured in a pure RTN by encoding more information into the states in a way analogous to Gazdar's Generalized Phrase Structure Grammar (GPSG) [12], though we believe that it is more efficient to factor these different components and thus reduce the number of states by a vary large factor.

3.1 Features

For example, we could encode agreement facts by exploding the nonterminal set to distinguish features such as person, number and gender so there would be 3*2*3 types of nouns, one for each combination of features. Similarly, there would be an equal number of verbs, one for each of the agreement possibilities. Though these numbers are finite, they are certainly rather large; 10^5 was an estimate given by Kurt Konolige in a talk at the 1980 meeting of the ACL. This appears to be a particularly inefficient representation of features; a more efficient representation would capture the fact that features are strongly constrained objects which can be manipulated in just a few highly restricted ways.

We will represent features as a separate component from the part of speech. Each feature is another field in the chart, so now the chart contains n-tuples of the form: $\langle s, i, j, f_1, f_2, \ldots \rangle$ to represent a phrase of category *s* spanning from position *i* to position *j* with features f_1, f_2, ... (We will occasionally use the term *record* for these n-tuples because they are actually implemented as records.) The multiplication rule is also modified to manipulate features in the appropriate way. For example, the rule that multiplies an NP with a VP to form an S will check the features of the two daughters for agreement and it will assign the appropriate features to the result.

3.1.1 Overriding Features in Exceptional Cases

Certain types of features tend to have idiomatic exceptions which a parser has to be able to deal with. For example, certain verbs like *die* and *sneeze* are generally intransitive although they can take on transitive interpretations under certain very restrictive idiomatic conditions such as:

(50) John sneezed a big sneeze.

(51) John died a horrible death.

The lexical entry for the verb *die* declares it to be intransitive except for this one case, which is marked by placing a property on *die* to override the intransitive subcategorization when there is a noun phrase whose head is the word *death*. In order to make this check more efficient, we have added the *head* slot to phrases in the chart, so that it is relatively inexpensive to find the head of a phrase. The exception handling mechanism has been generalized to accept other subcategorization violations such as:

(52) They are [on board [the ship]] now.
(53) They are [on top [of it]] now.
(54) They are [within [5 miles] [of the ship]].

In these cases, a preposition is taking a second argument, either a NP as in (52) or a PP as in (54). The former case is handled by adding a special phrase structure rule: PP → P N NP with the condition that the N transition is permitted only if the P (*on* in this case) is marked for the particular head (*board* in this example). In a case such as (54), we suppress the issue to semantics since *5 miles of the ship* will be accepted by the existing syntactic component, though it will have a slightly different structure, namely: [within [[5 miles] [of the ship]]]. In semantics, we raise *of the ship* to *within*. With an exception handling mechanism of this sort, it is much easier to express the facts; without such a mechanism it would be almost impossible to impose any subcategorization restrictions because there is almost always an exception of some sort or another.

3.1.2 Representation of Features

There are a few engineering issues that enter into the representation of features in the chart. We have previously noted that the chart has random access on *s* and sequential access first on *j* and then on *i*. How should we represent features? We have chosen to store one computer word full of features for each combination of <s, i, j>. This is a somewhat arbitrary engineering decision, though it does have some interesting implications. First of all, the multiplication rule will first generate states and then filter out bad states using the features. One could imagine an alternative representation scheme where the features are used in the generation process and the states are used in the filtering.

Secondly, we have very little space to represent features so we will have to decide carefully which ones should be represented at this level and which ones can wait for semantic interpretation. We have concentrated on the better discriminators, those features that weed out the most number of combinations; it isn't worthwhile representing a feature at this level if it will almost always allow the multiplication to succeed.

The third consequence is somewhat more difficult to deal with, though one can imagine some possible improvements which are probably worth exploring. Suppose there were two phrases that were identical to each other in every way except for their features. Our scheme will represent both of them with a single set of features, though ideally they should each have their own. Consider the following example below where there are two ways to form an NP over the same words with different values for the number feature.

(55) [$_{NP}$ Flying planes] make too much noise. *plural*
(56) [$_{NP}$ [$_{VP}$ Flying planes]] is very dangerous. *singular*

Our representation will say that these two phrases are merged in a single n-tuple (record) which is marked both singular and plural. That is, we will *merge* the two phrases together into a single record with the *union* of the features. This is a bit too loose because sometimes it will include some combinations that don't exist. For example, there is no way to represent the fact that (57) has only two interpretations, not four. That is, the subject and the predicate can be either singular or plural, but they both have to agree.

(57) Flying planes can be dangerous. *ambiguous*

Nevertheless, we have found this representation to be extremely effective and efficient. In practice this union of features rarely causes us to over-accept. It may be reasonable occasionally to accept too many parses at this level of processing and filter them out at a later level.

3.1.3 The Features in EQSP

EQSP currently uses 37 features. Since our PDP-10 computer has 36-bit words, most features are represented with a single bit, though two of them share the same bit. Most of the features are used for subcategorization; there are very few agreement features because there is very little room and because agreement features are relatively poor discriminators. Only 6 of the 37 features are used for agreement: *first-person-singular, third-person-singular, plural, present-tense, past-tense* and *plural-object*. It was decided that other agreement features such as gender and case do not weed out enough combinations to justify keeping them in the feature vector at this early stage of processing.

3.1.3.1 Subcategories

There are a number of features which are used to further subdivide certain parts of speech in order to restrict and/or expand their usage. There are five features in this class: *time-phrase, place-phrase, pronoun, numeral,* and *converted-participle-adjective*. The first two are good examples of how features can expand the usage of a part of speech. These have three additional usages beyond those of a normal noun phrase. They can be used adverbially as in (58), in post-modifier position as in (59), and they can conjoin with semantically similar prepositional phrases as in (60).

(58) I gave Tom a present yesterday. *Adverbial*
She must have done it next door.
*I gave Tom a present John.
*She must have done it John.

(59) The concert yesterday was very exciting. *Post-Modifier*
The party next door was very exciting.
*The concert John was very exciting.
*The party John was very exciting.

(60) We'll do it tomorrow and in the month of May. *Conjoined with PP*
She must have done it either next door or at work.
We will be having a sale the rest of this week and in the month of May.

In contrast to the *time-phrase* and *place-phrase* features which expand the use of noun phrases, the *pronoun* feature restricts the use of noun phrases. There are certain noun phrase positions which exclude pronouns:

(61) *I picked up it.
I picked up John.

(62) *Here comes it.
Here comes John.

(63) *I saw him who you like.
I saw the boy who you like.

(64) *I saw [him from England].
I saw [the boy from England].

(65) *It was him two.[20]
It was plant two.

---

20. There is an exception for *you* and *we* which can take post-modifiers in a sentence like: *You two seem to be having a good time*

The numeral feature is used to allow sentences such as (66) where there is no punctuation to signal conjunction.

(66) Give results for plants 1 2 3 4.

A *converted-participle-adjective* is used in a heuristic which avoids constructing adjective phrases such as (67) when there is also participle reading as in (68). The AP parse will be constructed just in case there isn't a corresponding participle phrase. This happens in two cases. First, there are words like very which block the participle interpretation as in (69), and secondly, there are certain adjectives like *fond* and *angry* which have no participle counter-part. We use the feature *converted-participle-adjective* to distinguish words like *fond* or *angry* which are only adjectives, from words like *persuaded* which are both participles and adjectives.

(67) I was [$_{AP}$ persuaded]. *adjective*

(68) I was [$_{VP}$ persuaded]. *participle*

(69) I was very persuaded. *"very" selects for adjectives*

(70) I was fond of candy. *No Participle Counter-Part*
I was angry at you.

3.1.3.2 Subcategorization

Most of the features are used for syntactic subcategorization, which seems to be a particularly good discriminator.

feature	example
verb-is-intransitive	John died.
verb-takes-sentential-first-object	John thinks that I did it.
verb-takes-sentential-second-object	John persuaded Bill that I did it.
takes-bare-infinitive	John made them take the exam.
verb-takes-at-most-one-np-after-it	*John forced Bill Sam
particle-possible	John put the book down.
verb-doesnt-take-of	*I [put [the picture] [of him into the box]]
factive	the fact that John did it; *the task that John did it

There are two kinds of semantic subcategorization. First, there is a notion of a responsibility-center, an object that can own things like a person or a company. It is a very useful distinction to make in the management domain in which we are concentrating our efforts. There are two features for this notion

which have the obvious interpretation: *first-of-two-nps-after-verb-must-be-a-responsibility-center* and *responsibility-center*. The second type of semantic subcategorization is for unknown words as in sentences like:

(71) Define the term "tsum" to be the total sum of all products.

There are two features for parsing these sorts of constructions: *introduces-unknown-word* and *quote-head-noun*. The former is assigned to words like *define* (as in *define top to be*) and *let*; the latter is used for nouns like *word* (as in *the word foop is*), *term*, and *phrase*.

3.1.4 Transformational Context

It is useful to have a few features for keeping track of the context for certain "transformations." (We are using the term "transformation" somewhat loosely since most of them are implemented with a base generation approach.) These divide into three types: top level contexts, unbounded contexts, and adjunct contexts. Many linguistic constructions are sensitive to these different contexts. For example, there is a class of so-called *root transformations* (eg. auxiliary inversion, and imperative deletion) which occur only at top level, and there is a class of so-called *unbounded transformations* (eg. wh-movement and conjunction) which apply across several clauses subject to certain very strong linguistic constraints (contexts). There is considerably less linguistic interest in the third class, though we have found it to be very useful to be able to filter out certain pre- and post-modifiers. We will discuss these features in more detail when we introduce the relevant transformations.

(72) top level structure: question-sentence, declarative-sentence, presentative-sentence[21]

(73) Wh-movement and Conjunction: wh-must-be-used, sequence-going, that-or-which-comp, in-question, relative

(74) adjunct contexts: noun-phrase-rejects-post-modifiers, phrase-is-post-modified, noun-phrase-has-numeral-modifier, noun-phrase-is-post-modified-by-clause, comma-appositive, premodifier-needed

3.1.5 Adjunct Contexts

Adjunct contexts turn out to be fairly important since they are extremely common and hence it is worthwhile to spend a little more effort in this area of the grammar. We have introduced several features to deal with pre- and post-modifiers. These are important because the parser can almost

---

21 The term *presentative* is a generalization of *imperative* following Joos [16].

always find a potential modifier and it will significantly reduce the number of final parses if just a few of these possibilities can be excluded. The simplest example is that pronouns usually reject post-modifiers as in:[22]

(75) *He from England ...
The boy from England ...

(76) *He that you like ...
The boy that you like ...

(77) *He 5 is selling well.
Product 5 is selling well.

The feature *premodifier-needed* is used to accept (78), while rejecting (79).

(78) the big, red ball

(79) the big, ball

It is turned on after a comma and turned off when a following modifier is found. This is an example of the point made by the Thompsons [7] that features can be used to avoid a further articulation of the grammar rules themselves. The table below provides a quick summary of the features used to reduce the possibilities of modification.

feature	accepts	rejects
premodifier-needed	the big, red ball	*the big, ball
noun-phrase-rejects-post-modifiers	item from England	*he from England
noun-phrase-has-numeral-modifier	*2 dogs	2 3 dogs
phrase-is-post-modified	item 5	*[item from plant] 5
comma-appositive	John, the baptist	*John, the baptist, the great phophet

There are some undergeneration problems with these features. For example, the *post-modifier* feature will exclude *Jack in the box 1* because it will treat *in the box* as a PP, rather than as part of an idiomatic noun. Similarly, the feature *noun-phrase-has-numeral-modifier* will reject the phrase: 2 3 *legged dogs*. Fortunately, neither of these cases appear in the MALHOTRA Corpus.

In summary, we have introduced features to eliminate ungrammatical parses and also to exclude extremely rare possibilities. It is much more efficient to undergenerate at the syntactic stage, and if

---

22 There is a proverbial meaning of *he* which permits relative clauses as in *He who walks under tall ladder will find himself in a heap of trouble.*

the parser should miss the appropriate parse, there would be a special procedure to backtrack over the chart and recover the appropriate interpretations on a second pass. The alternative requires the parser to carry along at each point every interpretation that could possibly be correct in some completely wild and unlikely semantic context.

3.2 NP-Movement

NP-movement covers a wide range of linguistic phenomena including passive, there-insertion, and raising. All of these transformations are relatively easy to process compared with wh-movement, because they do not nest and hence they can all be performed locally within a lexical entry (as in Lexical Functional Grammars [19]) or within a constituent (as in Phrase Structure Grammars [PSG]). Our analysis is essentially identical to the lexical based generation approach currently advocated by Bresnan and Kaplan [19]; the morphology routine decides which lexical entries are applicable and assigns the subcategorization features appropriately. Predicate argument relations are determined in another pass which combines additional subcategorization facts with semantics and domain restricted pragmatics. This pass will be the subject of a forthcoming paper.

3.3 Wh-Movement

Wh-movement is one of the hardest transformations to parse because it is the source of considerable ambiguity as we have seen with reduced relative clauses. Much of the psycholinguistic literature attributes the problem to one of finding empty categories; there is an implication that wh-movement would be much easier if the gaps were identified by some lexical object like the word *gap* or a resumptive pronoun. Though this is correct, we have found that it is also difficult to locate the filler. Consider a sentence like (80) where there is only one possible gap, but there are quite a number of possible fillers.

(80) I saw the block in the corner of the kitchen near the yard that you liked.

One might think that such sentences are quite rare, but this has not been our experience. Consider the following sentences taken from the MALHOTRA corpus [22] which illustrate that fillers can be quite ambiguous.

(81) What are the components of the various costs you know about?

(82) Print every piece of information you have concerning plant 0 in 1972 and 1973.

It would be very inefficient to "move" all possible fillers into all possible gaps. It would be a very serious mistake to enumerate the sub-trees for each possible filler times the number of possible gaps because the number of sub-trees grows with the Catalan numbers. EQSP attempts to merge a number of fillers together and move them all at once. For example in (81), EQSP will discover that all noun phrases ending at the word *costs* are possible fillers and move them in one step downward until it finds a gap in the embedded relative *you know about*. Then it checks to see which of the possible filler-gap pairs are possible.

There is another serious source of inefficiency; it is important to avoiding looking for gaps where there aren't any fillers because it is almost always possible to find a gap if you look hard enough. This is a particularly easy mistake to make in a partially bottom-up parser like ours because the obvious implementation would tend to find gaps first and then check to see if there are any fillers. The solution to this problem is very similar to the *usefulness* notion introduced in the first chapter; the parser should only look for gaps if they might be useful, that is, only if they are in the context of a filler.

There are many ways to implement wh-movement subject to these efficiency considerations. For example, one could encode the relevant information into the non-terminals directly, as Gazdar does [12], and then apply the general context-free algorithm to the resulting grammar. We have chosen a slightly different approach for EQSP because we believe that his requires too many non-terminals, though it is worthwhile to present his approach as a starting point for discussing our own solution.

3.3.1 Gazdar's Formulation of Wh-movement

Gazdar encodes the necessary information into the nonterminals by enlarging the class of nonterminals to include a number of *derived categories* which are like normal categories except that they dominate a gap. So for example, he would have a derived category for an S with an NP gap (written S/NP),[23] and a category for a VP with an NP gap (VP/NP), and a category for an S with a PP gap (S/PP), and so on. A trace of an NP is written NP/NP. He shows that it is possible, with these derived categories, to write context-free rules for wh-movement constructions. We have given the analysis of a simple example below.

---

23 This notation is different from the slash convention used earlier in this paper and it is also different from categorial grammars [2].

Grammar Rule	Example
NP → NP S/NP	$[_{NP}$ [the boy] $[_{S/NP}$ you like]
S/NP → NP VP/NP	$[_{S/NP}$ [you] $[_{VP/NP}$ like]]
VP/NP → V NP/NP	$[_{VP/NP}$ [like] $[_{NP/NP}$]]

Gazdar's proposal is currently a very exciting and controversial area of linguistic inquiry. Most of the debate hinges on whether or not this system is an adequate model of linguistic competence. It is unclear how such a model could handle multiple extractions from a single constituent or how it could handle crossed movements, both of which are rare in English though one can find some possible examples. We will give two examples of multiple extractions, the second of which appears to be crossed.

(83) Which violins$_i$ are these sonatas$_j$ easy [to play t_j on t_i?]
(84) What$_i$ don't you know [how$_j$ to solve t_i t_j?]]

These are also somewhat problematic for our approach, as we will see. Currently we don't handle form (83) though we could extend our system to handle it by stacking wh-fillers or by treating t_j as noun phrase movement recovered in semantics. Form (84) is parsed by raising *how* to *know*, as in *know the method to solve*.

Though these issues are extremely interesting and important, their current resolution is beyond the scope of this paper; we will avoid questions of linguistic competence and concentrate on processing considerations for the time being. Assuming that Gazdar's model is linguistically realistic, how does it fare on efficiency considerations? It has the obvious drawback that it approximately squares the number of categories without taking advantage of a number of constraints on wh-movement. This was the same drawback that led us to reject his proposal of encoding features directly into the nonterminals; just as we believe that it is more efficient to keep category information separate from features, it is probably more efficient to separate category and wh-movement. However, before rejecting his proposal, we should point out that it fares quite well on the two points mentioned above, merging multiple fillers and finding just useful gaps. We will adopt a modified solution which captures most of the benefits of Gazdar's analysis without exploding the space of nonterminals. First, consider the problem of multiple fillers as in the following example:

(85) Here is the block on the table in the kitchen on the first floor that you like.

This example is ambiguous because the gap could refer to any of the noun phrases. It is important that the wh-movement transformation does not enumerate all possible fillers. Notice that Gazdar's proposal fares well in this respect because it analyzes the gapped clause: $[_{S/NP}$ you like] just once, even though there are quite a number of noun phrases that the gap might refer to.

Finally consider the problem of useless gaps. Again Gazdar's proposal performs well because the previous implementation of the usefulness notion has exactly the desired behavior when applied to Gazdar's derived categories. That is, the algorithm avoids looking for useless gaps because those derived categories will not be marked useful without first finding a filler. (Fillers precede and command their gaps in leftward wh-movement.)[24]

3.3.2 Wh-movement in EQSP

Our solution is designed to achieve the virtues of Gazdar's proposal without exploding the space of categories. The idea is similar to the approach taken for features; at a very global level, it is a fairly old notion that has appeared in many systems [4, 18]. We will add another slot to the chart which holds the "slashed" category; let us call this the *wh-sent* slot to suggest the ATN notion of *send* arcs. This requires a slight modification to the multiplication rule and the usefulness procedure which are fairly straightforward. The multiplication rule carries the wh-sent value down from a mother to one of its two daughters. (Alternatively, one can think of it carrying the value up from one of the daughters to its mother.) The usefulness procedure has to mark daughters with the desired wh-sent value. We will not discuss these details here.

This system works fairly well though it is often important to know something about the internal structure of the filler. For example, it is often useful to know about the features of the filler and therefore the wh-sent slot should contain more information than just the category of the filler. The following example illustrates the need for propagating number features along with wh-movement.

(86) What is happening?
(87) *What are happening?

EQSP uses these agreement features to find filler-gap relations in sentences like (88)-(89). The first is using number features to distinguish between the complementizer use of *that* and the pronominal

---

24 This argument does not apply for rightward movement rules such as heavy NP shift or wh-movement in Turkish. In these cases, the algorithm will look for useless gaps but it won't look for useless fillers.

use. The latter is using number features to decide whether the verb *to be* is inverted or not.[25]

(88) I saw the girls$_i$ that e_i are changing. *complementizer that*
I saw the girls that$_i$ is changing e_i. *pronominal that*
(89) What are the results *e*? *inverted be*
What is (e) the result (e)? *ambiguous be*

This sort of feature propagation is automatic in Gazdar's framework because his categories already encode all the features, though it is somewhat more difficult for us because we have already factored the features out of the nonterminals. We could also incorporate the features into the wh-sent slot, though we have chosen to pass a pointer to the filler itself.[26] This doesn't change the usefulness condition, though it does complicate the multiplication rule. It is important to merge several fillers together so that the parser finds the gaps just once, rather then once for each possible filler. This is accomplished by filling the wh-sent slot with a list of possible fillers. For example, given a structure like (90) below, the wh-sent slot would contain a list of noun phrases ending just before the word *that*. This is actually implemented as a pointer into the chart, so it does not require too much additional storage.

(90) Here is the block on the table ... on the first floor that you like.

We can manipulate this list as a unit without ever enumerating all its members. For example, we can check agreement constraints by looking at the top level features of the fillers; it isn't necessary to look deep down inside the fillers. This is an important theoretical point because there are the Catalan number of subtrees ending at a given point, but only a linear number of top level phrases.

3.3.3 Adjacent Filler-Gaps: A Special Case

Consider examples like (91) where the filler and gap are unambiguous and adjacent. In this special case, it is somewhat inefficient to use the general wh-movement mechanism which is fairly expensive. Consequently, EQSP has a special case to look for these obligatory subject gaps. When EQSP finds such a gap, it removes gap-finding from the useful things to do in order to block the normal gap finding mechanism from coming into play and looking for a gap anyway. This turns out to be an extremely useful special case because it applies to a very large percentage of the MALHOTRA Corpus.

---

25 In fact, EQSP canonicalizes the inverted and uninverted interpretations of sentence (89). The two interpretations can be paraphrased as *What is it that the result is?* and *What is it that is the result?*

26 This may well take the generative capacity outside the class of context-free grammars.

(91) What is near ...

(92) What is going ...

(93) The girl who is going ...

3.3.4 Complement Clauses, Relative Clauses and Questions

There are several different types of wh-movement; these distinctions turn out to be very useful because they can constrain the possibilities of wh-movement which is a relatively expensive operation. There is a three way split between complements (94), relative clauses (95), and questions (96).

(94) It is true that John is a nice guy. *complement*

(95) I saw a man who is a nice guy. *relative clause*

(96) What is true? *question*

In certain cases, the parser can determine which case is which by looking at the wh-word (eg. *that, who, what*). That is, certain wh-words induce certain types of clauses and not others. For example, *how*, *what* and the determiner use of *which* do not start relative clauses or complement phrases:

(97) *It is true how John is a nice guy.

(98) *I saw a man what is a nice guy.

This constraint is implemented in EQSP by assigning each wh-word a feature for each type of wh-clause that it can begin. So for example, the word *how* is assigned a feature for introducing questions (called *in-question*), but not one for relative clauses (called *relative*).

This is another important lexical clue that distinguishes a minimal pair such as (99)-(100). The second can be parsed as a complement clause or as a relative clause. The pair are contrasting the lexical entries of *girl* and *report*; the latter optionally selects complement phrases unlike the former. This distinction is represented in EQSP through the *factive* feature. In this way, EQSP can make the grammatical distinctions necessary in order to find two parses for (100) and just one for (99).

(99) I saw that girls that Bob changed. *relative clause*

(100) I saw the report that Bob changed. *ambiguous*

It has also been noticed that relative clauses will rarely center-embed more than two times. This turns out to be a very useful filter for excluding certain possibilities of relative clauses.

(101) # [The man [who the boy [who the students recognized] pointed out] is a friend of mine.]

There are no cases in the MALHOTRA Corpus with more than one level of center-embedding, and hence we block spurious analyses by excluding two deep center-embedding from the grammar. The feature *noun-phrase-is-post-modified-by-clause* is set inside a relative clause so that EQSP will not look for another modifying clause.

3.4 Conjunction

Conjunction is perhaps the single most difficult construction to parse. It has all of the combinatoric problems associated with prepositional phrase attachment (unbounded direct ambiguity) plus the problems associated with transformations of finding dependencies between fillers and gaps. We will begin with a relatively simple general purpose mechanism and then we will complicated it with special case modifications as they become necessary. Originally we had hoped that the general purpose mechanism would cover almost all the cases, but experience seems to indicate that this is not possible, at least with current linguistic understanding and current engineering technology. Our implementation is sufficiently general to cover all cases of conjunction in the MALHOTRA corpus, though it is far from the complete and definitive solution to conjunction.

3.4.1 The General Mechanism

Suppose we introduced a number of rules into the grammar as follows:

(102) NP → NP CONJ NP
VP → VP CONJ VP
S → S CONJ S
PP → PP CONJ PP
⋮

Now the normal context-free parsing algorithm would find most of the easy cases. Note that this mechanism has two drawbacks: it is both incomplete and inefficient.

3.4.1.1 Coverage

Let us address the completeness issue first and then we will return to efficiency considerations. This approach does not yet account for a wide range of linguistic transformations associated with conjunction such as: VP-deletion, subject deletion, gapping, right node raising and the across the-board convention. It seems that we will have to complicate the mechanism to account for these facts

in any case.

Linguistic Constraint	Examples
VP-deletion	Jack saw Bill and Sam did *e* too.
subject deletion	Jack saw Bill and *e* hit Sue.
gapping	Jack saw Bill and Sam *e* Sue.
right node raising	John gave Mary *e*, and Joan presented *e* to Fred, books which looked remarkably similar.
Across-the-Board	The kennel which [Mary made *e*] and [Fido sleeps in *e*] has been stolen. *What fork did John give Mary a knife and *e*. *What knife did John give Mary *e* and a knife.

There are numerous ways to complicate the grammar in order to accommodate these facts. For example, Gazdar [11] proposes that one can capture the across-the-board facts by encoding the wh-filler information into the nonterminal (as he normally does for wh-movement) and then restricting the conjunction mechanism to identical nonterminals. We use a similar technique, though slightly modified because we represent the filler in the *wh-sent*, rather than encoding it into the nonterminal. Consequently, we capture the across-the-board facts by insisting that both constituents have the same *wh-sent* values.

3.4.2 Idiosyncratic Cases

One could imagine similar proposals for the other transformations. We have basically adopted this general approach, though we have chosen to implement it in a more efficient way that avoids exploding the space of nonterminals. However, there is another class of problems that we have noticed in practice which have not attracted much attention in theoretical linguistics. It appears that the word *and* can be deleted in certain restricted cases such as:

(103) What was the total cost of raw material in 71, 72, 73?
(104) What was the total cost of raw material in 71 72 73?

This is very interesting from a computational point of view because it means that the conjunction cannot always depend on the the presence of words like *and*. Note that almost every conjunction mechanism has this basic flaw; they all trigger on a conjunction word like *and*. It would be an extremely serious mistake to allow the conjunction markers to drop out uniformly in all cases because

this would find quite a number of absurd parses, especially if one considered interactions with the other deletion rules mentioned above. This kind of example led us to believe that some of these deletion rules have to be considered on a case by case basis. They occur where there is a strong semantic context. For example, we will only allow the word *and* to delete in a sequence of several "semantically similar" noun-phrases such as a sequence of dates as above, or in a phrase like: *items 1 2 3 4*.

Actually the problem may be much more serious than this. One can argue that almost any syntactic generalization involving conjunction can be overridden in a strong semantic context. For example, there is a rather strong syntactic generalization that conjunction applies to two constituents of the same part of speech. However, there are some possible counter-examples to this; time, place and manner adverbs can conjoin with time, place and manner prepositional phrases as in sentence (105) below. We have implemented an exception to the general mechanism to handle cases such as these.

(105) We expect difficulties [$_{ADV}$ now] and [$_{PP}$ in the future].

This we allow using the feature time-phrase. Similarly, it is tempting to claim that conjunction applies to complete constituents, not fragments, though again there are numerous possible counter-examples where the syntactician would be forced to resort to a deletion analysis. The gapping sentences mentioned above are good examples; we've included two more cases below:

(106) John [drove through *e*] and [completely demolished *e*] a plate glass window.
(107) Mary expressed [costs] [in dollars] and *e* [weights] [in pounds].

The first of these we currently don't handle. The second we do handle by taking all conjunction as between strings of constituents in a phrase rather than between constituents, and then restricting the strings to length 1 except in special places such as just after the verb. In some sense, a constituent with a hole in it is a partial constituent, and so in this view, the deletion analysis is a way of conjoining partial constituents, at least in some cases. Unfortunately it is very difficult to process deleted/partial constituents because there are so many combinatoric possibilities. One seems forced to say that deletion rules are greatly restricted by semantic constraints, and then one is left with almost no useful syntactic constraints. If conjunction really is as bad as all this, and we certainly hope that it is not, then it is hard to imagine how one could do better than a case by case analysis. We have attempted to use the general mechanism as much as possible, though we have taken some liberties in certain restricted cases in order to combat the combinatoric nature of the problem.

3.4.3 Conjunction and the Size of the Grammar

Even if we were able to remove all the problems associated with deletion rules and other semantically idiosyncratic cases, Gazdar's approach still leads to a radical increase in the size of the grammar because every part of speech can potentially conjoin with another constituent of the same part of speech and hence we have introduced a new grammar rule for every part of speech. Though this is already very expensive, it gets significantly worse if we reintroduce deletion rules. It appears that context-free grammar rules are a very inefficient way to represent conjunction; we would really like to parse the meta-rule "XP → XP and XP," instead of expanding the meta-rule out for each possible part of speech. We have essentially implemented this by adding a general purpose conjunction mechanism that looks for states (a) ending just before the word *and*, and (b) initiated by the first word after the conjunction. It starts these states after the word *and*. So for example, it would start an NP and S after the word *and* in (108) below because each of these categories end just before the word *and* and each can begin with the word *the*. (Condition (b) is a classic use of lookahead.)

(108) [[The robot] [put [the block] [on [the box]]]] and the ...

This is fairly easy to implement in terms of dotted-rules and the chart; we will not go into the details here. In this way, we have reduced the size of the grammar in a significant way. When EQSP believes that is has reached the end of the second conjunct, it checks that the two conjuncts are "syntactically parallel." This is a somewhat complicated test that basically requires both conjuncts to have the same part of speech and the same wh-gaps and so on, though there are some further restrictions and some exceptions. In particular, there is an exception which allows time and place phrases to conjoin even if they have different parts of speech, as mentioned previously. There is also a restriction for numerals so that they don't conjoin with non-numerals and in: **John and 2*, except after *for* as in: *for 1971 and plant 2*. Similarly, there are some restrictions for deletion rules so that both conjuncts have the same number of constituents. Top-level clauses have some additional restrictions. For example, EQSP accepts (109) but not (110). These restrictions are expressed in terms of the top-level context features: *question-sentence*, *declarative-sentence*, and *presentative-sentence*.

(109) Who broke in here and what did he want?
(110) *Who broke in here and I don't like it.

There is a very important added constraint on combinations of three or more conjuncts which blocks arbitrary nesting as in (111). If we allowed all possible nestings, there would be the Catalan number of possible parses. In fact, we limit the depth to 2, so there are the Fibonacci number of possible parses. This number was empirically determined and will be justified in the next section, which discusses

empirical results.

(111) # ... [products and [products and [products and products] and products] and products] ...

Multiple conjuncts are somewhat easier to parse when there are commas instead of *and*s. In this case, EQSP uses the *sequence-going* feature to distinguish (112) from (113) at the point where the *and* is reached. In the first case (112) conjunction has already been initiated by the comma while in the second case (113) it hasn't.

(112) x, y and z
(113) x and z

In summary, we employ three types of conjunction mechanism.

(114) Top level:	What did he do and why did he do it?
(115) General:	What did Bob hit and kick?
	Express weight in pounds and height in feet.
(116) Context determined special cases:	Items 1 2 3 and 4.
	Results in 71 72 73.

4. Experimental Results

EQSP has been tested on the MALHOTRA Corpus [22] (appendix I) and the LADDER-TODS Collection [15] (appendix II). Three statistics were kept: sentence length, number of parses and cpu time. Theoretically, the time should be cubic (or better) with sentence-length and the number of parses should be Catalan (or better) with sentence-length. Unfortunately, it is somewhat difficult to test these predictions directly on the two Corpuses because there are too many interacting constructions such as wh-movement, prepositional phrases and conjunction which combine in complicated ways that make it very difficult to predict exactly how the three variables will be related. We have attempted to isolate these various factors by constructing sequences of synthetic sentences such as (117) and (118) which exercise a particular construction, and in this way we were able to compare various constructions with each other and with the two corpuses. These two synthetic series are particularly interesting because they completely bracket all of the sentences in the MALHOTRA Corpus. The possessive noun phrases are faster per word than any sentence in the MALHOTRA Corpus, and the gerunds are slower. See the plot below showing the synthetic points completely enveloping the MALHOTRA Corpus. It was expected that possessive noun phrases would be fast because they are LR(0), but the gerund result was somewhat surprising; we had expected conjunction or prepositional

phrases to be the worst case. We will return to these examples shortly.

(117) It was margin's margin. *best case*
It was margin's margin's margin.
It was margin's margin's margin's margin.
⋮

(118) What was involving? *worst case*
What was involving involving?[27]
What was involving involving involving?
⋮

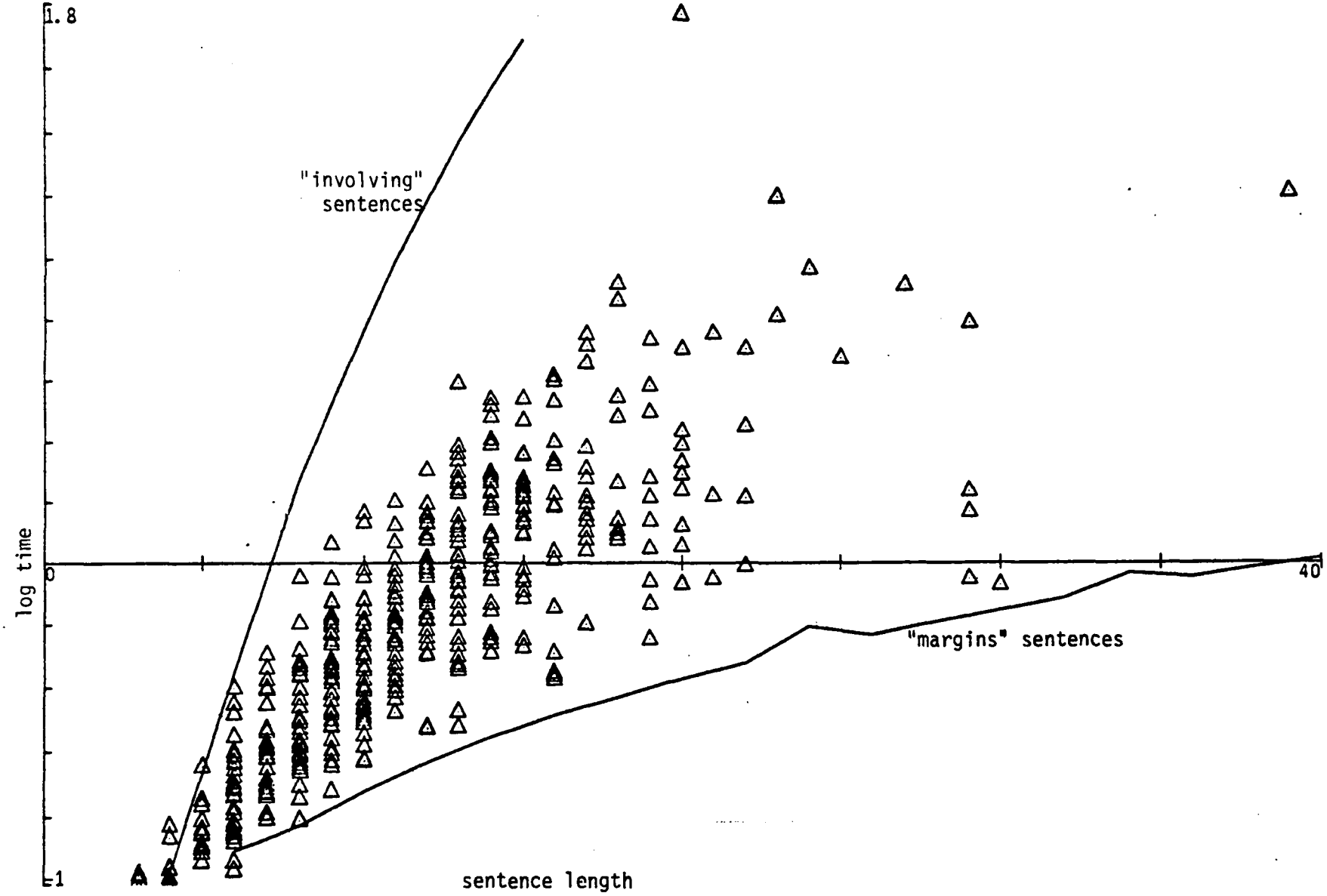
These synthetic sentences have led to some other very interesting observations. The number of parses forms a mathematical series which we have been able to solve in certain cases, so that it is possible to predict the exact number of parses for arbitrarily long sentences. Even though we have a theoretical account for this series, it is quite remarkable that, even in practice, it describes the number of parses *exactly*, despite all of the tuning that goes into a large engineering project of this sort. The Catalan series is one such example; we have also constructed synthetic sequences which generate products of two Catalan numbers and the Fibonacci [20] numbers. These will be presented in turn.

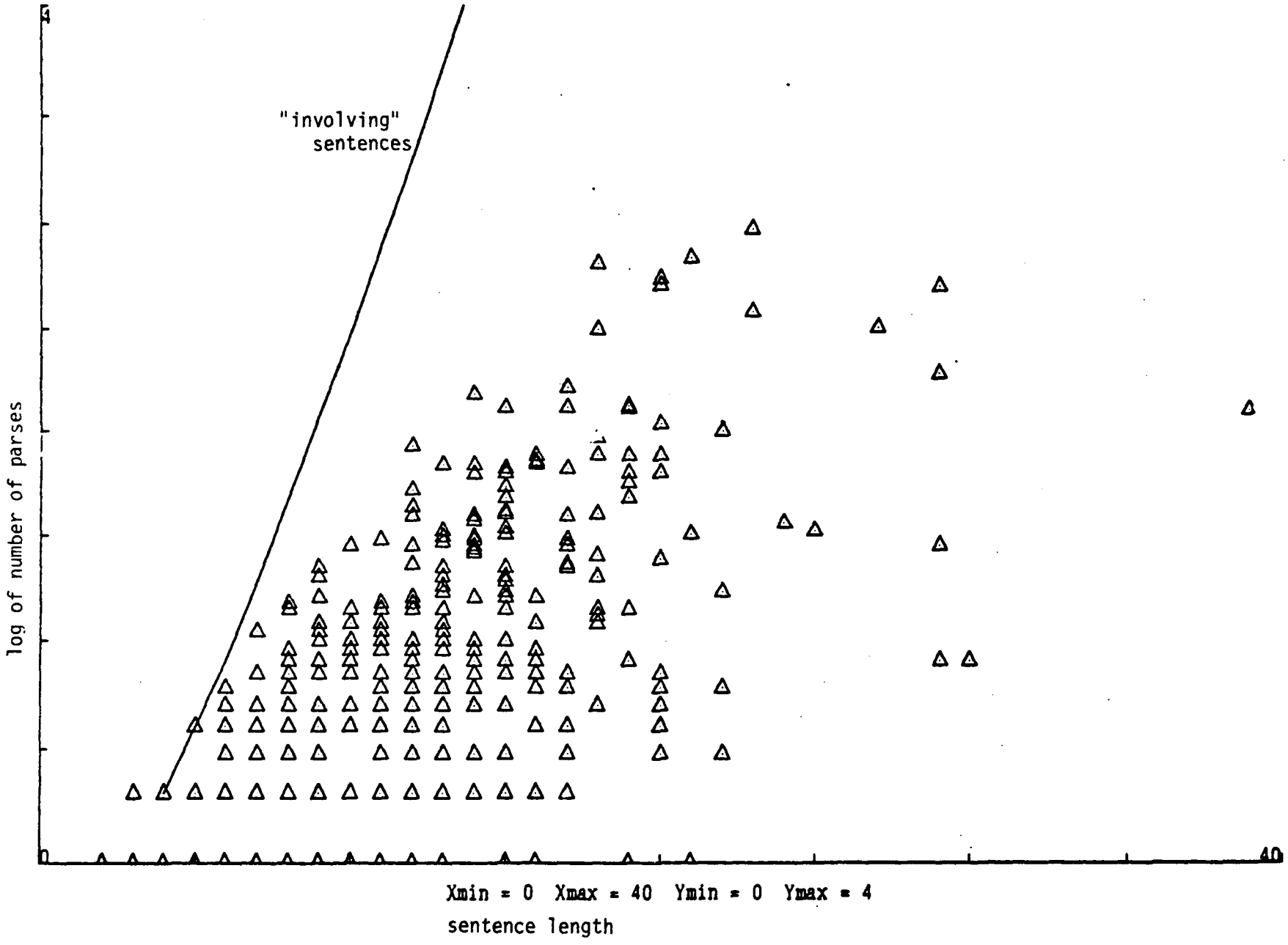
One might wonder why we should look for these series. They indicate a fundamental computational process which describes the search space at a very abstract level. For example, the Catalan series suggests that all possible parses are grammatical and hence the parser is not excluding much of anything; the parser is enumerating all n^3 points in <i,j,k>-space, and in this way, it is following a grammar very similar to 'Grammar AA'. This claim can be made without looking inside the parser to see if it is top-down, bottom-up, left-to-right, or whatever. Similarly, the product of two Catalans suggest two independent subgrammars each of which behave like 'Grammar AA'. The Fibonacci series suggests yet another class of abstract computations with certain other well-known characterizations.

Nevertheless, many of these series grow extremely quickly; it may be impossible to enumerate these numbers beyond the first few values. And hence, it can be argued that it isn't worthwhile to make distinctions between them; some of them are prohibitive and some of them are worse than prohibitive. From an engineering point of view, it doesn't make sense to distinguish between the Fibonacci series and the Catalan series for large numbers; in either case, it will be impossible even to print all the parse

---

27. EQSP has an *ing ing* filter which blocks many of possibilities, but it does not block the gerund and adjunct interpretations. Perhaps the *ing ing* filter should be more general.


1.8
"involving"
sentences
log time
0
40
"margins" sentences
-1
sentence length
Xmin = 0 Xmax = 40 Ymin = -1 Ymax = 1.8


"involving"
sentences
log of number of parses
0
40
Xmin = 0 Xmax = 40 Ymin = 0 Ymax = 4
sentence length

trees, and harder still to perform semantic interpretation. Therefore, the argument goes, it must be necessary to introduce some techniques for shrinking the search space such as canonicalization, semantic constraints, determinism, etc. Although we are extremely sympathetic with some or all of these techniques, we believe that it is worthwhile to establish what will happen *without* them. With this baseline, it becomes much easier to evaluate a particular technique. Presumably the more useful ones will drastically reduce the number of parses from the baseline, and less useful ones will have little effect.

Finally, it is worthwhile to look at these series carefully because some of them are tractable over practical ranges and some of them are not. For example, the Fibonacci series remains tractable over a much larger range than the Catalan does, and hence if we can show that most of the practical examples lie within the tractable range for a Fibonacci process, but not for a Catalan process, then it would be a real accomplishment to reduce the search space from the Catalan numbers to the Fibonacci, as we have done for conjunction.

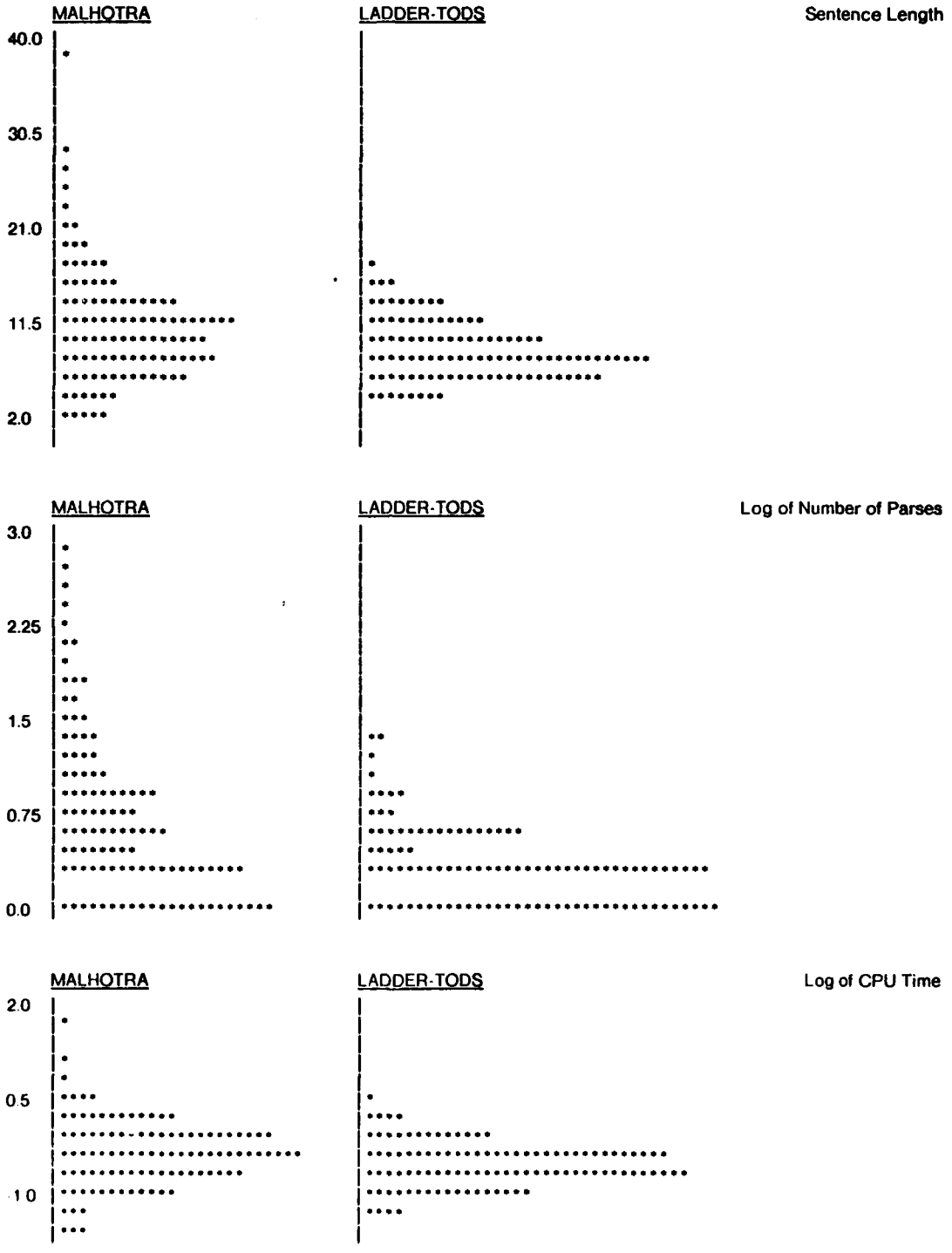
4.1 A Comparison of the LADDER and MALHOTRA Corpuses

It appears that the LADDER-TODS Collection is considerably easier on the whole; the sentences are shorter, they have fewer parses, and they take less time. Furthermore, the LADDER-TODS Collection is more compact; the standard deviations are smaller for each of the three statistics: sentence length, number of parses, and cpu time in seconds. The histograms are given in figure 3 and a table is given below showing the means, standard deviations, minimums, medians, and maximums. The histograms of number of parses and cpu time are plotted on a log scale in order to bring in a few points with very large values. The data have been normalized to account for the different sizes of the corpuses. Each point on the MALHOTRA plots corresponds to five sentences, and each point on the LADDER-TODS plots corresponds to a single sentence.[28] The higher number in the table below represents the MALHOTRA Corpus and the lower number represents the LADDER-TODS Collection. It is particularly interesting that the medians for number of parses and time are much smaller than the means; this suggests that most of the data are well behaved and just a few extreme points are causing most of the trouble.

---

28 Round off errors were taken to the next higher integer. That is, the last point on the MALHOTRA plots corresponds to between 1 and 5 sentences.

Fig. 3. Histograms of MALHOTRA and LADDER Corpuses



	Mean	Standard Deviation	Minimum	Median	Maximum
Sentence Length:	11.2	5.1	2	11	39
	9.3	3.0	4	9	18
Number of Parses:	22.5	81.1	0	4	958
	3.3	4.5	1	2	30
CPU Time:	1.2	3.1	.0	.6	58.5
	.5	.5	.1	.3	4.1

The differences between the two corpuses may be due to the ways in which they were collected. Ashok Malhotra gathered his sentences by fooling a number of management experts into believing that he had solved the natural language problem. They were then asked to try out his new program, called the Perfect System. In fact, his program was a fake; it actually connected the subject directly to an experimenter sitting in the next room, and in this way he was able to gather an interesting set of sentences which may be fairly representative of a typical query in the management domain. This approach works very well with computer naive domain experts:

> "In fact, surprising as it may seem, few subjects realized that the experimenter was *creating the responses until they were* told so after the experiment. Until this secret was revealed, many subjects were extremely impressed by the range of capabilities displayed by the system. Thus, the Perfect System could be said to be a success as the subjects behaved as if it were an ideal English language question-answer system." [22: pp. 57]

On the other hand, the LADDER subjects had a more realistic expectation of the machine's capabilities, and consequently, they may have adjusted their responses appropriately. The LADDER Group has noted that their sentences seemed somewhat biased, though they offer another possible account:

> "Fortunately, our users have been very helpful by tending to avoid the use of the long and complex constructions that are most likely to lead to ambiguities. Perhaps this is because the teletype medium inclines users to prefer short, simple constructions." [15: pp. 133]

Both LADDER and EQSP perform very well on the LADDER-TODS Collection. It is difficult to compare EQSP with LADDER without more data. The EQSP results are given in Appendix II.

4.2 Synthetic Sentences

As mentioned above, it is very difficult to compare parsers with these corpuses because there are too many different factors averaged together. We have explored a new technique for isolating particular factors. In this way, we were able to make some concrete predictions. First we will discuss the number of parses and then cpu time.

4.2.1 Catalan Numbers

As we have mentioned before, the number of parses is perfectly described by the Catalan Numbers for the following series of sentences:

1	It was the number of products?
2	It was the number of products of products?
5	It was the number of products of products of products?
14	It was the number of products of products of products of products?
42	It was the number of products of products of products of products of products?
132	It was the number of products of products of products of products of products of products?
429	It was the number of products of products of products of products of products of products of products?
1430	It was the number of products of products of products of products of products of products of products of products?
	⋮

We have also determined that the number of parses is exactly the product of two Catalans for structures like (119) and (120). This is to be expected because the two ambiguities are completely independent. Structure (120) is a very good example illustrating that gap finding is not the only problem for wh-movement; there is only one possible gap (in subject position), but there are the Catalan number of possible fillers.

(119) The number of products of products ... of products was the number of products of products ... of products.

(120) What number of products of products ... of products was the number of products of products ... of products?

This result seems to be very stable; it works for many prepositions and content words. (One might have suspected some variation due to lexical ambiguity.)

4.2.2 Fibonacci Numbers

The number of parses is exactly predicted by the Fibonacci numbers for the following series:

3	It was actual products and actual products and actual products.
5	It was actual products and actual products and actual products and actual products.
8	It was actual products and actual products and actual products and actual products and actual products.
13	It was actual products and actual products and actual products and actual products and actual products and actual products.
21	It was actual products and actual products and actual products and actual products and actual products and actual products and actual products.
34	It was actual products and actual products and actual products and actual products and actual products and actual products and actual products and actual products.
55	It was actual products and actual products and actual products and actual products and actual products and actual products and actual products and actual products and actual products.
89	It was actual products and actual products and actual products and actual products and actual products *and actual* products and actual products and actual products and actual products and actual products and actual products.

The next sentence in the series requires too much memory, though by Engineer's Induction, the next sentence ought to generate 55 + 89 parses. In this case, lexical ambiguity has been carefully controlled; the word *products* must be a noun and not an adjective or a verb which have different morphology (ie. *producing* and *produce*). We find a very different series if we replace the word *products* with the word *costs*, which can be construed as a verb as in: *That costs too much.* In this case, the series is: 3, 7, 15, 30, 58, 109, ... Unfortunately, we have not solved this series yet. Also, it should be noted that the word *actual* is playing a crucial role; without it, EQSP will find exactly one parse no matter how many conjuncts there are.

Where does the Fibonacci series come from? Previously, we noted that conjunction ought to grow with the Catalan numbers. However, there is a performance constraint in EQSP that blocks arbitrary nesting such as:

(121) # ... [actual [products and actual [products and [actual products]]]]

The maximum depth in conjunction is set to 2, as noted near the end of the conjunction section. That is, the n^{th} conjunct can conjoin at its current level or it can conjoin one level higher. The Prepositional phrases differ in that attachment is possible at any depth, not just 0 or 1, and hence they generate the Catalan number of parses. With this restriction on conjunction, there are the Fibonacci number of parses, because the number of ways to conjoin n conjuncts is the sum of the number of ways to conjoin the conjuncts at the current level plus the number of way to conjoin those at the embedded level. How many conjuncts can be at each level? There are at most n – 1 at the current level and at most n – 2 at the level before that. Therefore, the number of parses of n conjuncts is the sum of the number of parses of n – 1 and the number of n – 2 which is the Fibonacci series. In the lexically unambiguous case, the initial case (n < 2) is 1 and hence we get the familiar numbers. Notice that this analysis fails with lexical ambiguity for two reasons. First the initial cases have several parses and secondly, the inductive cases can combine in more ways than just the sum of the two parts because these ambiguities trigger deletion possibilities and multiple grammar rules, etc.

4.2.3 Worst Case for Number of Parses

As we have mentioned before, the worst case that we have found is (122), which generate more parses per word than any sentence in either corpus as illustrated in the plot above. It can be argued that many of the parses are ungrammatical because the *ing-ing* filter should be stronger than the one in EQSP. However, we have noticed that (123) are almost as bad, and there is no infinitive-infinitive filter in English (though there is one in Italian). EQSP's *ing-ing* filter excludes *ing* participles as verbal complements to an *ing* participle phrase, though it allows them as gerunds and adjuncts. The gerund and adjunct possibilities alone would make these sentences worse than the prepositional phrases sentences because there would be the combination of two Catalan series, not just one. However, there are some more ambiguities. First, the wh-movement can go almost anywhere and secondly, the auxiliary *was* can invert around any string of *involving*s. Finally, *involving* also has an adjectival part of speech, so there are quite a number of other parses where a few of the *involving*s are taken as prenominal adjectives modifying a gerund.

(122) What was involving involving involving involving ...
(123) What was to total to total to total to total ...

It is a very interesting fact that there will be many fewer parses if we insert a closed class word such as *of*, *with* or even *and* after the first few *involving*s because it restricts wh-movement, inversion, and lexical ambiguity.

4.3 Analysis of CPU Time

As we have noted, in chart parsing, grammars fall into one of three classes: time n, time n^2 and time n^3. We have found the same three classes, though the times appear to be slowed down by a factor of n, due to the representation of the chart which is optimized for average case, not worst case. Recall how we organized the chart. For each state, there is a list of records containing *i* and *j* and other useful information such as features, wh-sent, conjunction, and so on. These lists are sorted on *j* and then on *i* to improve average time performance. It is also assumed that these lists don't tend to be very long because sentences tend to vary constructions, so that it is unlikely that the same state will be used very often in the same sentence. However, this assumption is incorrect in these synthetic sentences where we have intentionally used the same state again and again. In these cases, it can take n^2 time to search down the list of records attached to that state in order to compute chart(s,?i,j), whereas a matrix representation could find the answer in n time by enumerating each i and checking that square in the matrix. Hence, this algorithm has a worst case behavior of time n beyond that of Earley's algorithm. However it is much better than Earley's algorithm when the lists are short because it can compute chart(s,?i,j) in constant time, whereas a matrix representation would require linear time. The fact that our representation is more efficient when constructions are varied is compatible with the intuition that sentences are easier for people to understand under the same conditions. It is certainly true that these synthetic sentences are extremely unnatural; it is hard to imagine a context where one would utter such a sentence even with more meaningful content words.

However, it is somewhat difficult to verify time bounds directly by running the parser and measuring time versus length because several factors interfere with the time measure such as paging and garbage collection. Although we have attempted to factor these out, it is extremely difficult to remove all of them. Hence we have taken an alternative approach of inserting a counter into the inner-most loop that searches down the list of records, and in this way we have computed the exact number of records that are examined. It appears that the parser spends almost all of its time searching down records and hence this statistic, henceforth called the record count, is a good idealization of cpu time. In fact it is almost linear with time, though it is impossible to find an exact mathematical relation because the cpu time measure is somewhat noisy for reasons mentioned above and for a few other small factors such as a linear overhead for reading the input words and looking them up in the lexicon. Henceforth we will use record count instead of time because it is easier to work with. One record count corresponds to a time unit on the order of a millisecond, though it can vary by an order

of magnitude as we will see.[29] Record count is a fairly good replacement for cpu time for most of the synthetic sentences, though it is better for simpler sentences like possessive noun phrases (124) and worse for more complex conjunction (127) where relatively more time is spent in heuristic sections of the parser which are not measured by the record counter. We have plotted the record count against time for four sets of synthetic sentences, progressing from simpler constructions to more complex. The slopes are: 1.6, 5.4, .7 and 21.4 milliseconds per record traversed. The important point is that the lines are almost linear and that the slopes are on the order of a millisecond; we have no explanation for the variability in slopes or the non-linear residues.[30]

(124) It was margin's margin's ... margin's margin.
(125) He believes he believes he believes ... he believes it.
(126) It was products with products with products ... with products.
(127) It was costs and costs and costs and costs ... and costs.

Record count turned out to be a very useful abstraction because, with the aid of MACSYMA [24], we were able to find an exact mathematical relation for record count in terms of sentence length. By substituting time for record count, we have an approximate relation for cpu time in terms of sentence length. We then performed regression analysis to find the constants in the relations. The fits were extremely encouraging (R-square = .9999). These results suggest that (128) is $O(n^2)$, (129) is $O(n^3)$ and (130) is $O(n^4)$. (The function *odd* distinguishes the odd and even length sentences; it returns 1 if sentence length is odd and, 0 otherwise. This term is usually important for sentences with inversion; we have no explanation for it in this case.)

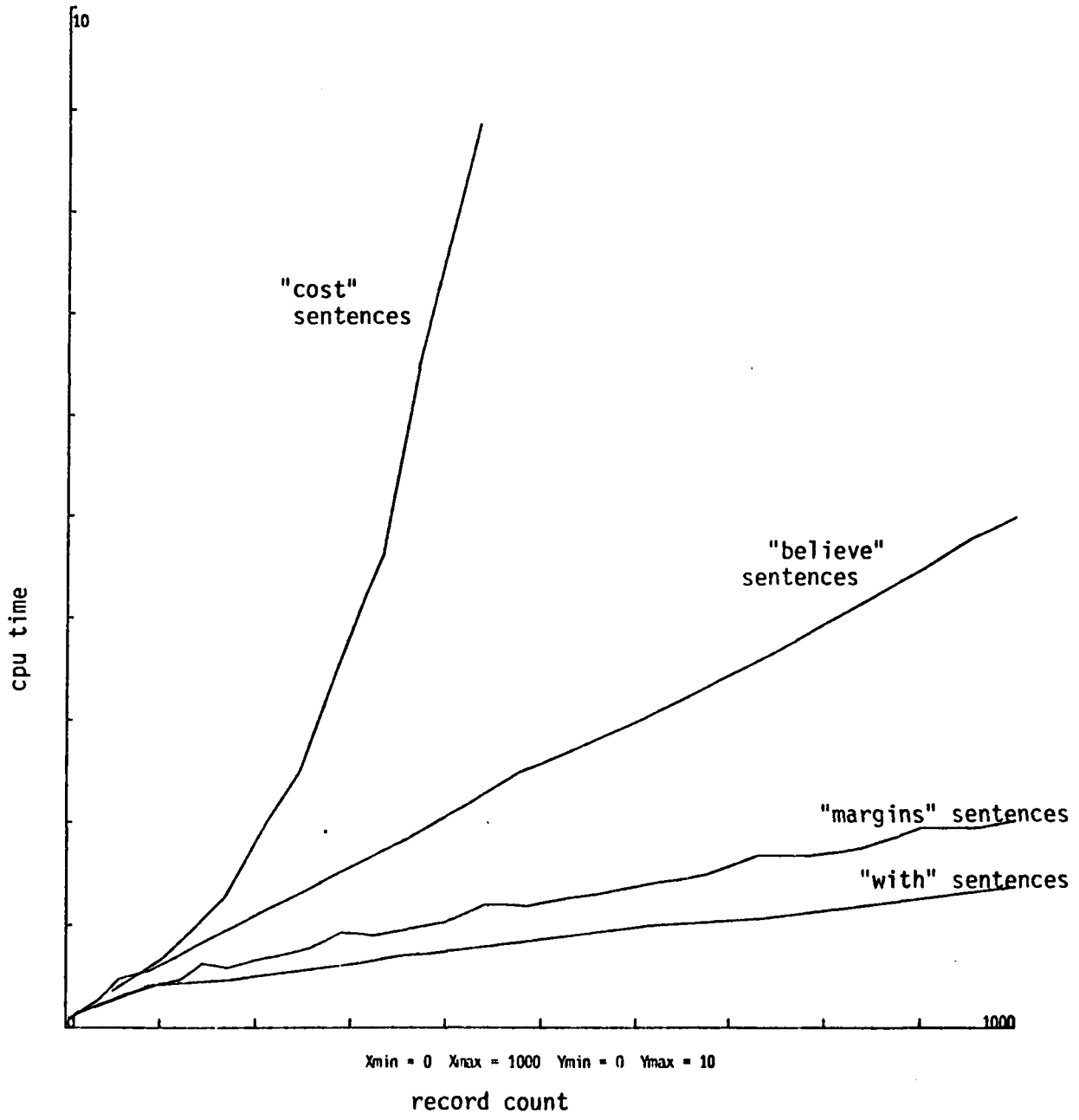
Sentences	Record Count as a function of Sentence Length
(128) It was margin's margin's margin's ...	$\frac{1}{4}n^2$
(129) He believes he believes he believes ...	$\frac{1}{24}n^3 - \frac{1}{4}n^2 + \frac{1}{6}n - 1$
(130) It was products with products with ...	$\frac{1}{96}n^4 - \frac{1}{16}n^3 + \frac{5}{24}n^2 - \frac{19}{16}n + \frac{209}{32} + \frac{1}{2}\text{odd}(n)$

These three cases are very similar to the three classes of grammars for Earley's Algorithm, except that ours are slower (in the worst case) by a factor of n. For example, (130) is similar to 'Grammar AA' in

---

29 Also, much of this time (perhaps a half millisecond) can be attributed to the computation involved in incrementing the record counter itself because of a technical error in the implementation of the counter. A millisecond is approximately 500 machine instructions on our KA PDP-10.

30 These lines were fitted with r-squares of .9960, .9994, .9995, and .9533, respectively. We have much less confidence that the last line is linear because a plot of the residues shows a strong parasitic effect. This could be explained by showing that the conjunction mechanism is taking a considerable amount of time which seems to be the case, though we have not explored this hypothesis adequately.


10
"cost"
sentences
"believe"
sentences
"margins" sentences
"with" sentences
cpu time
1000
Xmin = 0 Xmax = 1000 Ymin = 0 Ymax = 10
record count

that it is every way ambiguous. This is known to require n^3 time on Earley's Algorithm, and because our algorithm is slower by a factor of n, it requires time n^4 on EQSP. The table below compares the three synthetic cases above with grammars that behave similarly on Earley's Algorithm (with no lookahead).

Grammar	Synthetic Sentences	Time on Earley's Algorithm
$A \rightarrow Aa \mid a$	It was the margin's margin's ...	n
$A \rightarrow aA \mid a$	He believes he believes ...	n^2
$A \rightarrow AA \mid a$	It was products with products with ...	n^3

The analogy for the first and third case should be fairly clear. The first case is a left-branching structure where the usefulness condition (Earley's Predict) works very well and hence the chart has only O(n) entries, as opposed to the normal $O(n^2)$. This also brings the time bounds down to time n for Earley, though we need time n^2 because all n records happen to be associated with the same state in the grammar and hence we have to look down all of them in order to find chart(s,0,j). Similar comments hold for the third case, which is the worst case for both algorithms, though the constants are significantly worst in the *involving* sentences mentioned above where there is more lexical ambiguity.[31] These constructions are every way ambiguous and hence Earley's Algorithm will have to enumerate every point in <i,j,k>-space. EQSP has the same problem with these constructions, not surprisingly, though it has the additional problem that all the records happen to be attached to the same state so it requires another factor of n.

The *believe* sentences are somewhat more interesting because they illustrate a case where lookahead would be very helpful. With lookahead there are only n entries in the chart, but without lookahead there are n^2 entries because the parser can't know which rule to expand, unless it can lookahead to see if there is another input afterwards.[32] If there is another input token, then it should expand the recursive rule ($A \rightarrow aA$), and otherwise, it should expand the terminal rule ($A \rightarrow a$). In this case the parser knows exactly what to do at each point, and hence there will be only n records in the chart. But if the parser didn't look ahead, it would have to expand both rules, and consequently there will be n^2 records in the chart. As we mentioned in section 2, the lookahead condition has the same effect on right branching grammars that usefulness has on left branching grammars; that is, it reduces space bounds from n^2 to n by making the parser behave deterministically. This also improves the time bounds by a factor of n.

---

31. The actual relation for the *involving* sentences is: $n^4 - \frac{31}{3}n^3 + \frac{61}{2}n^2 - \frac{163}{6}n + 55$.

32. In other words, 'Grammar aA' is not LR(0), but it is LR(k). Therefore, it will behave deterministic if the parser looks ahead, but otherwise, it will not.

In summary, we have shown that cpu time grows with n^2, n^3, and n^4 depending on the grammar in much the same way that Earley's Algorithm depends on the grammar. We have also shown that the number of parses can grow as fast as the Catalan numbers. Finally, we have found a best case and a worst case which bracket the two corpuses in both cpu time and number of parses. This provides an attractive empirical verification of the theoretic predictions (of time and space bounds) discussed in the first two sections. In addition, we have found these synthetic sentences to be a very effective tool in analyzing the parser's performance.

5. Conclusion

Perhaps the conclusion is a good place to summarize some of the strengths, limitations, and uncertainties we believe would be encountered in proceeding to further develop the EQSP parser. As strengths we can cite the parse time and the coverage.

It is now clear that if a parser is written in compiled rather than interpreted form it will be fast enough for many practical purposes on the bulk of sentences it receives. This has already been shown for parsers which find only one parse or have a limited grammar [4]. We have shown that it also holds for a parser which finds all parses in one of the more complete grammars extant. We have, in fact, shown that this is true if only syntactic constraints are used. Some people state that parse time is no longer critical, but for reading large volumes of text (eg. proof reading), it is still an issue.

With regard to coverage, careful study of the well known English grammar books, linguists' examples, and corpuses of actual input has led us to the informal conclusion that our current EQSP system covers a major portion of the syntactic constructions one would want in a parser. Discounting the lexicon, the EQSP system occupies about 87 pages of LISP code and took about 10 man months to program. The transition network defining the grammar has about 131 nodes. This is a small program in comparison with other systems which have been built in LISP. There seems to be no question that we can extend the syntax as far as necessary without running into the so called complexity barrier, where the program becomes so complex that it cannot be kept coherent using current programming tools.

There are, of course, some potential problems with EQSP which further work may or may not resolve. For one, all our constraints are hard and fast rules; they are either met or they are not. This means that there are four sentences in the MALHOTRA corpus which we currently reject as syntactically unacceptable.

(131) *What would have 1972 prices have been?

(132) Product 4 and 5 show greatest variance, why?

(133) Give me two tables, the contribution margin in each plant for the years 1972 and 1973.

(134) Which years do you have costs figures for?

In each of these the offending element is underlined. The first of these exhibits a serious confusion of two constructions. The remaining three involve problems of agreement. In the second, a singular noun is in apposition with two product names. In the third, a plural noun *tables* is in apposition with a singular noun *margin*. The construction involving *margin* is, however, semantically plural because two years are given. In the fourth a plural noun has been used as a modifier of a head noun. This is a very rare construction. It occurs with pluralia tantum, eg. *scissors blades*, but not often with other nouns. We were willing to make *sales* a pluralia tantum to handle *sales figures* because one could possibly say, *sales is the most important thing*. But to make *costs* a pluralia tantum seems wrong. The fact is that plural modifiers are syntactically acceptable, although rare. As we further extend the grammar to rare constructions, more situations of this type will undoubtably arise.

We could accept these sentences by loosening our constraints, but then many extraneous parses of other sentences would be found. Our plan is to build a second diagnostic pass which operates when no acceptable parses were found on the first pass. The first pass will rule out rare constructions which produce many false parses. It will also rule out grammatical errors. This two pass plan will only work if, when the intended parse is so ruled out, no other acceptable parse remains. This is true for the sentences we have tried but we don't know if it will be true in almost all cases. Perhaps when such a situation arises people would also be fooled. However, as we stated above, they have better semantics and pragmatics than we can mount in the short run.

Another problem we face is to reduce the parse time of long sentences full of explosive constructions. Surely, applying all semantics and pragmatics as we go would greatly help, if not solve, this problem. However, our guess is that this will extend the parsing time for the typical sentence, where cheap syntactic constraints alone eliminate almost all parses and thus give semantics and pragmatics an easier job. One solution to this would be to use different strategies depending on the length of the sentence, but this would require us to have the sentence in hand before we begin parsing. We would prefer to parse the sentence as it is being typed, in an on-fine fashion, rather than waiting for the entire sentence to be input and only then decide what to do with it.

Our current thinking is to do part of the semantics as the sentence is parsed and the rest on a second pass. We feel that the semantics and pragmatics of noun phrases up to the head noun will be critical,

but this is just a guess. We will also apply predicability constraints in filling slots of a subordinate clause. Determining the correct use of PP's would be very helpful in reducing the number of parses, but we feel it may be too expensive for the first pass.

Yet another uncertainty arises when we have to apply semantics to a sentence which has a large number of syntactic parses. Using a chart naturally leads to combining all parses into one structure. Thus instead of having the two separate parses (135) we have the combined structure (136).

(135) This is $[_{NP}\ [_{A}$ flying$]\ [_{N}$ planes$]]$

This is $[_{NP}\ [_{VP}\ [_{V}$ flying$]\ [_{NP}$ planes$]]]$

(136) This is (OR $[_{NP}\ [_{A}$ flying$]\ [_{N}$ planes$]]$
$[_{NP}\ [_{VP}\ [_{V}$ flying$]\ [_{NP}$ planes$]]])$

We hope that the semantics module can cut off whole branches of combined structure, thus eliminating more than one parse at a time. This has been true for the limited semantics module now implemented. However this operates more at the level of LADDER-TODS rather than MALHOTRA. Going from one to the other greatly increased the demands in syntax and this might happen in semantics. We found it very complex to try to maintain a factored structure combining alternative semantic interpretations. This can perhaps be done, but for the moment we have opted to construct separate semantic interpretations. If there are a great many of these we are again faced with a blow up of required resources.

So far, we are pleased with the EQSP parser. The results in the appendices are very encouraging. Time will tell how well the above problems can be solved. If they can, we will have a very useful parser.

6. Acknowledgments

We would like to thank Glenn Burke and Peter Szolovits for their discussion, comments, and programming assistance both on this parser and its predecessors. Peter Szolovits also helped us proof read this final draft which was completed after the death of the first author.

References

1. Aho, Alfred V. & Ullman, Jeffrey D., *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, Inc., 1972.

2. Bar-Hillel, Y., Gaifman, C., and Shamir, E., *On Categorial and Phrase Structure Grammars*, The Bulletin of the Research Council of Israel, 9F,1-16.

3. Bresnan, J., *The Passive in Lexical Theory*, Occasional Paper #7, Center for Cognitive Science, 1980 and also to appear in Bresnan (ed), MIT Press, 1981.

4. Burton, R., *Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems*, BBN Report No. 3453, 1976.

5. Chomsky, N., *On Binding*, Linguistic Inquiry, 1980.

6. Church, K., *On Memory Limitations in Natural Language Processing*, MIT/LCS/TR-245, 1980.

7. Dostert, B., and Thompson, F., *How Features Resolve Syntactic Ambiguity*, in Proceedings of the Symposium on Information Storage and Retreival, Minker, J. and Rosenfeld, S. (ed), 1971.

8. Earley, Jay, *An Efficient Context-Free Parsing Algorithm*, Unpublished Ph.D Thesis, CMU, 1968.

9. Earley, J., *An Efficient Context-Free Parsing Algorithm*, Communications of the ACM, Volume 13, Number 2, February, 1970.

10. Ford, M., Bresnan, J., and Kaplan, R., *A Competence-Based Theory of Syntactic Closure*, paper presented at the Sloan Workshop on Parsing Long Distance Dependencies at University of Massachusetts at Amherst, 1981 and also to appear in Bresnan (ed), MIT Press, 1981.

11. Gazdar, G., *Unbounded Dependencies and Coordinate Structure*, to appear in LI, 12.2, 1981.

12. Gazdar, G. *Phrase Structure Grammar*, to appear in P. Jacobson & G. Pullum (eds.) *The Nature of Syntactic Representation.*

13. Graham, S., Harrison, M. and Ruzzo, W., *An Improved Context-Free Recognizer*, ACM Transactions on Programming Languages and Systems, pp. 415-462, Vol 2, No. 3, July 1980.

14. Harris, L., *Experience with ROBOT in 12 Commercial Natural Language Data Base Query Applications*, pp. 365, IJCAI-79.

15. Hendrix, G., Sacerdoti, E., Sagalowicz, D., and Slocum, J., *Developing a Natural Language Interface to Complex Data*, ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978, pp. 105-147.

16. Joos, M., *The English Verb: Form & Meanings*, The University of Wisconsin Press, Madison, Milwaukee, and London, 1968.

17. Kaplan, R., *Augmented Transition Networks as Psychological Models of Sentence Comprehension*, Artificial Intelligence, 3, 77-100, 1972.

18. Kaplan, R., *A General Syntactic Processor* in *Natural Language Processing*, Rustin R. (ed.), Algorithmics Press, 1973.

19. Kaplan, R., and Bresnan, J., *Lexical-Functional Grammar: A Formal System for Grammatical Representation*, Occasional Paper, Center for Cognitive Science, 1980 and also to appear in Bresnan (ed), MIT Press, 1981.

20. Knuth, D., *Fundamental Algorithms*, Vol 1 in *The Art of Computer Programming*, Addison Wesley, 1975.

21. Kuno, Susumu, and Oettinger, A. G., *Multiple Path Syntactic Analyzer*, in *Information Processing*, North-Holland Publishing Co., Amsterdam, 1963.

22. Malhotra, A., *Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis*, MIT/LCS/TR-146, 1975.

23. Marcus, M., *A Theory of Syntactic Recognition for Natural Language*, MIT Press, 1980.

24. Mathlab Group, *Macsyma Reference Manual*, Laboratory for Computer Science, MIT, 1977.

25. Milne, R., *A Framework for Deterministic Parsing Using Syntax and Semantics*, DAI Working Paper 64, Department of Artificial Intelligence, University of Edinburgh, 1980.

26. Pratt, V. R., *A Linguistics Oriented Programming Language* IJCAI-3, 1973.

27. Pratt, V., *Lingol - A Progress Report*, IJCAI-4, 1975.

28. Ruzzo, Walter L., *General Context-Free Language Recognition*, unpublished PhD Thesis, University of California, Berkeley, 1978.

29. Sheil, B., *Observations on Context-Free Parsing*, Statistical Methods in Linguistics, 71-109, 1976.

References

30. Shipman, D. and Marcus, M., *Towards Minimal Data Structures for Deterministic Parsing*, IJCAI79, 1979.

31. Steele, G., *The Definition and Implementation of a Computer Programming Language Based on Constraints*, MIT, AI-TR-595, 1980.

32. Valient, L., *General context free recognition in Less Than Cubic Time*, J. Computer and System Sciences 10, pp. 308-315, 1975.

33. Woods, W., *Transition Network Grammars for Natural Language Analysis*, Communications of the ACM, Volume 13, Number 10, October, 1970.

Appendix I - Results with the MALHOTRA Corpus

The first number is the number of parses and the second is cpu time in seconds on the MIT-ML Machine, which is roughly a half MIPS (million instructions per second) KA PDP-10 machine. The symbol " – " is used when EQSP was unable to produce results in a reasonable amount of time and space.

12	3.4	What was the percentage of overhead costs to total sales for the last five years?
2	1.2	What were the profit margins for each product for the last five years?
14	0.9	What are the overall profits on operations for the past 5 years?
2	0.7	What were the gross sales figures for the past five years?
152	2.4	What are profit margins as a percentage of sales for each manufacturing installation?
5	0.7	What is the profit contribution of each manufacturing installation?
1	0.0	OK.
6	0.9	What data do you have on operations as a percentage of gross sales?
3	0.1	Yes and for each plant.
1	0.0	Can you calculate percentages?
14	0.6	What are the production costs as a percentage of sales?
7	0.5	What are the deviations of production cost from actual?
1	0.1	Cancel this question.
16	1.7	What is the ratio of actual cost to budgeted cost for each product?
10	1.5	What is the percentage change in sales for each product for the past year?
7	0.4	What are variable costs for manufacturing operations?
16	0.5	Have transportation costs increased during the past years?
7	0.6	What are the ratios of production costs to sales?
6	1.0	What has the average selling price for each product been for the past two years?
12	1.4	What quantities were produced for each product for the past two years?
8	0.8	Please display overhead costs for all plants for 1972 and 1973.
7	0.6	What was budgeted overhead for all plants for 1972?
1	0.2	What were sales and profits for 1973?
1	0.1	Do you have variable budgets?
3	0.2	Do overhead costs vary with volume?
12	0.6	I believe your overhead variance accounts for your lower than expected profits.
4	0.2	I suppose I should.
2	0.9	What were contribution margins by product for 1972 and 1973?
2	0.2	How are contribution margins calculated?
3	0.7	What is the difference between list price and average selling price?
1	0.0	For 1972?
3	1.6	What is the difference between plant 1 and plant 3 and plant 2 and plant 4?
7	0.4	What are the components of overheads for the plants?
9	0.5	Is transportation cost part of operating expenses?
1	0.2	Where does transportation cost get included?
5	0.4	Give me the constituents of overheads for each plant?
2	0.2	I would like to end the interview.
5	1.2	List sales for product 1 through product 5 for the last two years?
85	2.0	List prices of single unit prices for both 72 and 73.
22	1.8	What was cost of producing each product for both 1972 and 1973?
2	0.3	Production cost first for one unit.
48	0.9	Did one plant assume more production of batteries from the other plants in 1973?
1	0.0	In 1972?
28	1.9	What was the rate of increase of shipping cost between 1972 and 1973?
6	0.6	Are shipping costs reflected in production costs?
2	0.3	Do you have any information on customer satisfaction?
3	2.0	What is the percentage of repeat customers in 1973 and 1972 and in 1971 and 1972?
1	0.3	What was the unit price in 1973?
63	2.4	What's rate of unit cost for each year and the ratio of this production increase to product price?
62	2.0	What is the percent of increase of each product for each year studied?
2	0.7	What was percent decrease in sales for last 5 years?
1	0.1	Cancel this question.
5	0.8	What was % of profit for each of last 5 years?
2	0.2	Increase over last year.
1	0.4	What were overhead costs for last 5 years?
5	1.3	What is list price vs selling price for last 2 years?
1	0.5	What's difference between list price and average quotation price?
0	0.6	Product 4 and 5 show greatest difference between list and quotation prices, why?
9	0.7	Do you have list of changes in sales force for each branch?
1	0.0	It should be included.

10	0.3	Give me a breakdown of items in your overhead.
3	0.2	Include each of these by plants.
2	0.3	Compare overhead costs for last 5 years?
9	0.7	Do you have further breakdowns of overhead attributable to each product within plants?
40	8.3	What % of each product is sold from each plant for each of the last 5 years?
7	1.6	What is total volume for each plant (sales) for each of the 5 years?
66	5.2	Compare plant overhead costs with total overhead costs for last 2 years, 1973 and 1972.
28	1.2	List increases in overhead for each plant for last five years?
5	0.4	Compare overhead costs for plants for last five years.
8	1.0	What are salary increases for each plant for last two years?
9	0.8	List increase in interest costs for last two years.
11	0.9	List inventory of product at end of 1971 and 1972.
3	0.4	List all data items you know about.
1	0.2	How many plants are there?
9	0.5	For 1973 list the sales of products by plant one, broken down by product.
5	1.4	For 1973 and for products 1 through 5 list prices and percentage profit margin.
8	1.5	For 1973 and plant 1 list direct manufacturing expenses by product and also total overhead.
5	0.3	Are the prices the same from plant to plant?
30	1.6	By plant, list overhead figures for 1972 and 1973.
16	1.5	For each plant, list the ratio of overhead to sales in 1972 and 1973.
–	–	In the future, please express numbers of over 100000 in terms of units of millions, and numbers over 100 but less than 100000 in units of thousands.
12	1.4	List production costs by plant for 1972 and 1973.
4	0.9	For each product, list the profit percentage for 1972 and 1973?
93	3.5	For each product list the ratio of total sales to total cost in 1972 and in 1973.
4	0.5	What is the definition of the profit for a product?
1	0.3	Were the prices the same in 1972 and 1973?
3	1.1	Why did you give me prices of $ 17, 18, 19.25, 20.25, and 18.0 earlier?
2	0.2	Do you have information about transportation cost?
2	0.9	What was transportation cost by plant for the last two years?
11	0.4	Is transportation cost included in overheads?
10	0.6	What are the components of the various costs you know about?
692	5.7	For each plant give the ratio of 1973 to 1972 figures for each type of production cost and overhead cost.
5	1.2	Why was there such a great increase in operating cost in plant 0?
40	2.3	Print every piece of information you have concerning plant 0 in 1972 and 1973.
2	1.2	Disregarding plant 0 totally, what is the difference in total profit between 1972 and 1973?
1	0.3	What is the difference in profit percentage?
3	1.1	What was the percentage increase in overhead costs between 1972 and 1973?
2	0.7	What was the percentage increase in the average price per product?
2	0.6	What was the average increase in the cost per product?
2	0.2	How did the product mix change?
2	1.0	What were the gross margin on each product in 1972 and 1973?
10	1.4	What is profit as a percent of sales in 1972 and 1973?
43	1.4	Are transportation costs included in overhead or cost of goods sold?
7	2.3	What components of the overhead costs go up more than 2% ?
10	1.8	What was overhead cost as a percent of sales in 1972 and 1973?
1	0.3	What was the increase in interest cost due to?
17	2.8	What would have 1973 profits have been compared to 1972 if the product mix had not changed in those two years?
6	1.0	How much did amount borrowed go up between 1972 and 1973 and how much did the average interest rate go up?
66	3.5	What percent of overhead cost is interest cost and what percentage is operating costs?
15	0.8	How much did operating costs go up between 1972 and 1973?
16	2.9	What were the five largest dollar increases in operating costs between 1972 and 1973?
14	1.3	How much was the dollar increase in operating costs and interest costs?
2	0.2	I know what the problem is.
2	0.6	What is the total sales for the past three years?
1	0.5	What was the net profit in 72 and 73?
2	1.2	What was the total cost of raw material in 71, 72, 73?
8	1.0	What was the dividend paid in 1971, 1972, 1973?
2	0.5	What if any are outstanding loans, 1971, 1972, 1973?
3	0.5	What was the interest rate, 1971, 1972, 1973?
2	0.1	What are the outstanding shares?
8	0.5	Any equipment purchased for long term depreciation?
3	0.4	Do you have information about what these loans are for?
2	0.3	Do you have labor cost for finished products?
2	0.9	What was the total labor cost for 1971, 1972, 1973?
4	0.2	Are we facing inflation?
4	0.4	What are we doing with the $16 million loan?
2	0.3	OK I think I know what the problem is.
1	0.3	What was total overhead in 1973?
0	0.3	Which years do you have costs figures for?
2	0.9	What were the overhead costs in each of the last five years?
4	0.9	What were total sales in each of the past five years?
1	0.3	What was the most profitable product in 1973?
1	0.0	Yes.

2	0.6	What were the profit margins on each product in 1972?
11	1.7	How much of each product was produced in 1972 and in 1973?
20	1.5	How much did the inventory level of each product change in 1973 from 1972?
3	0.8	What were the sales for each product in 1972 and in 1973?
2	0.6	What were profits in each of the last five years?
1	0.6	What were total sales in the last 5 years?
1	0.1	Cancel that question.
2	0.4	What were sales of each product in 1971?
7	1.9	What were the percentage increases in sales for each product in 1972 and 1973?
2	0.7	What is the overhead cost for each type of battery?
1	0.1	Overhead for 1972.
5	0.3	Difference in overhead from 1972 to 1973?
3	0.4	List the product mix for 1972 and 1973.
5	0.8	Give me the profit margin on each product for 1972 and 1973.
2	0.8	What is the cost of each product for 1972 and 1973?
3	0.8	What were sales for each product in 1972 and 1973?
4	0.5	Give me the budget for each plant and the overrun if any.
2	0.0	Production costs.
1	0.5	What is the overhead budget for each plant?
4	0.7	What overhead costs were incurred by each plant?
1	0.5	What is the percent overhead overrun at each plant?
33	2.2	Give me sales percent increase at each plant for 1973 over 1972.
1	0.2	What information do you have on competition?
3	0.3	Do you have any info on production costs?
34	1.4	Table direct cost, overhead cost, and contribution per unit sold for 1973.
1	0.1	How do you define margin?
36	9.3	Table sales in units sold, product mix, direct cost per unit and overhead cost per unit for the last 4 years?
5	2.7	Table unit sales, direct manufacturing cost, margin and product mix for the last 2 years by product.
1	0.1	Do you perform mathematical computations?
101	5.1	Please compute the following: percent change in unit sales, percent change in unit production cost from 1972 to 1973.
1	0.0	By product, please.
10	0.4	Do you have a forecasting model for demand?
2	0.2	Do you have any model at all?
1	0.2	List the functions you can perform.
19	1.6	Are there any variances between actual prices charged our customers and the guideline prices?
1	0.3	Please tabie them for product 4 for the five major customers.
10	0.9	List actual sales price and guideline price by product.
654	3.0	Do you have a model to maximize contribution to the company subject to production and other constraints?
10	0.9	Why were the quotation prices lower than list prices in 1973?
2	0.3	Have they been this way for the past years too?
1	0.0	No.
2	0.2	Please give me the overhead cost for 1972.
8	0.7	Table profit before tax for 1972 and 1973.
-	-	Compute profit for 1972 and 1973 according to the following formula: actual unit sales by product times list price minus production cost for the product summed over all products less overhead cost for the year.
2	0.2	I think I understand the problem.
1	0.0	Thank you.
1	0.6	What was the contribution margin of product 1 in 1973?
3	1.6	What were the actual and budgeted contribution margins of products 1, 2, 3, 4, 5 in 1973?
1	0.9	What were the contribution margins for products 1, 2, 3, 4, 5 in 1972?
5	0.4	Give the product mix in 1972 and 1973.
63	3.1	Give the actual and budgeted overhead costs in 1973 and the actual overheads in 1972 for each plant.
7	0.6	Give total contribution figure for 1972 and 1973.
1	0.0	Cancel.
7	0.6	Give total profit figure in 1973 and 1972.
12	0.7	Give list and actual prices for all products in 1973.
12	0.4	Give actual prices for all products in 1972.
3	0.3	Do you have a breakdown of overhead costs?
23	1.3	Give the breakdown of actual and budgeted overhead costs for plants 0, 2, 4.
556	58.5	Give actual and budgeted operating costs for all plants, and actual and budgeted management salaries and interest costs.
2	0.1	Give the budget profit.
2	0.2	Do you have data on transportation costs?
2	0.2	Do you have the data by plant?
4	0.4	Give budgeted and actual transportation cost by plant.
2	0.2	Give budgeted and actual sales revenue.
1	0.2	Give budgeted and actual inventory.
2	0.3	Give budgeted and actual selling costs.
1	0.2	How far back does your information go?
5	1.1	What was the % of overhead in each of the last five years?
2	0.2	Percent of overhead to sales.
28	1.9	Were there any changes in the product mix in terms of sales dollars?
2	1.2	What were the profit margins of the five batteries in the last two years?

14	1.1	What are the handling costs associated with each product and did they change over the last two years?
70	2.5	Handling costs are costs associated with products that are not reflected in direct MFG costs.
5	0.8	What are the actual selling prices of the five batteries?
48	1.6	How much was the additional revenue received from the 20% sales increase and where was it spent?
510	6.2	The intent of my question is to find out if you know if your accounting methods can relate the changes in sales to changes in your expense structures.
2	0.1	Does this help?
39	1.7	Please give me changes in each type of cost associated with each product.
958	6.5	In as much as allocating costs is a tough job I would like to have the total costs related to each product.
130	5.4	I mean I would like the cost of each product broken down on a direct and indirect basis.
2	0.8	What was the total production costs in the most recent two years?
1	0.7	Have any plants been supplying batteries to other than normal customers ie outside of their normal sales district?
20	1.2	Please display overhead figures (actual and budget) for all plants for the past four years.
12	1.7	Which plants were over budget on overhead by more than 5% ?
7	1.3	Please display the overhead budget variance in percent and absolute $ for plants 0, 2, and 4.
31	3.0	Which plants were over budget on fixed costs by more than 5% ?
15	1.2	Display the profitability of each plant divided by plant sales.
14	1.2	Display sales revenues for all plants for the past four years.
3	1.1	Display average company wide profitability for the last four years (%).
1	0.0	Yes.
4	0.9	Why is there such a difference between the company wide average profitability and the profitability of the independent plants?
4	0.3	I suggest we get rid of plant zero!
18	2.0	Has product mix changed in any plant whose profitability has fallen off?
165	5.7	Has product mix changed by more than 1% in any plant whose profitability has decreased?
8	0.9	Display the direct cost variance (absolute $ and %) for all plants.
4	0.6	Has there been a decrease in contribution margins for any product?
10	1.2	Display the percentage overhead growth for each plant for the past four years.
20	1.2	Display the overhead divided by sales (%) for each plant.
4	1.1	Why are the OH figures for plants 2 and 4 higher than for 1 and 3?
1	0.3	Has the profitability of any plant decreased?
1	0.2	Which one(s)?
6	0.7	Display the margins for plant 2 for the past 4 years.
322	8.2	Display the difference between list price and actual costs (direct + overhead) divided by list price for plant 2 for the past four years.
1	0.0	Yes.
5	0.6	Give me the breakdown of overhead expenses for the years 1972 and 1973.
16	1.0	Give me comparative numbers for operating costs for the years 1972 and 1973.
1	0.4	What was the profit margin for the year 1972?
1	0.0	Yes.
2	0.6	What was the sales revenue by product for the year 1972?
6	0.8	Give me the same revenue figures for the year 1973.
14	1.2	Give me the actual cost vs budgeted cost for each product for the years 1972 and 1973.
1	0.0	Yes.
25	3.0	Give me comparative figures for management salary, interest costs, and depreciation for 1972 and 1973.
1	0.5	What were the gross profit figures for the years 1972 and 1973?
5	1.3	What were the comparative figures for sales revenue vs direct costs for the years 1972 and 1973?
1	0.0	Yes.
76	1.9	Give me a breakdown of direct costs and overheads for each plant in the years 1972 and 1973.
76	1.9	Give me a breakdown of budgeted direct costs and overheads for each plant for the years 1972 and 1973.
2	0.9	Give me plant 0 production cost figure for the years 1972 and 1973.
41	2.3	By what percent did the overhead expenses in 1973 increase over those in 1972.
2	0.9	What were the comparative figures for overhead expenses for the years 1972 and 1973?
5	0.5	Give me details of how the additional sales revenue in 1973 was spent.
2	0.8	What was the product mix in the sales for the years 1972 and 1973?
1	0.4	What were the selling prices of each product?
10	1.1	What were average manufacturing costs for each product?
4	1.3	What were unit production costs for each product in the previous year?
8	2.5	What were the relative percentages sold of each product in 1972 and 1973?
4	1.3	What were average quotation prices for each product in 1972?
10	1.0	What were budgeted costs for each product in 1972 and 1973?
1	0.0	Both.
9	0.5	Do you have budgeted production costs on a per unit basis?
31	3.9	What quantity of product 1 was sold by all plants in 1973?
3	1.0	What were contribution margins for each product in 1972 and 1973?
31	1.9	What are the relative percentages of each product sold by each plant?
9	0.7	What are the relative percentages of sales by each plant?
2	0.3	Do you have list prices for each product?
1	0.2	What were they in 1972 and 1973?
512	5.1	Give me a breakdown of difference between list and average quoted price for each product for 1972 and 1973.
1	0.2	How many plants are there?
17	1.7	Which of the four plants had the largest value for total sales in 1973?
22	4.5	At plant 2, which product accounted for the largest percentage of total sales in dollars?
2	0.5	Does product 2 also account for the largest percentage at plant 4?

4	1.7	What was the total overhead of production for product 2 at plant 2 in 1973?
1	0.4	Substitute "direct manufacturing cost" for "overhead of production" in previous input.
382	15.6	What is the number of units of product 2 produced at plant 2 in 1973 times the unit cost of product 2?
1	0.3	Define the terms "unit cost" and "unit price".
382	15.7	What was the number of units of product 2 produced at plant 2 in 1973 times the unit price of product 2?
2	0.1	How is profit computed?
28	1.4	Can you produce a profit figure for a specific product at a specific plant in 1973?
312	7.3	Print a table containing unit cost and unit price for each product at plant 2 in 1973.
10	0.5	Compute unit cost for each of the products in 1972.
2	0.5	Which unit prices were different in 1972?
1	0.0	Print their values.
2	1.6	What were the total overhead costs at plant 2 in 1972 and 1973?
4	0.2	How is overhead cost computed?
1	0.2	List the fixed, non-manufacturing expenses.
108	1.7	For each of the factors just listed give the total value incurred at plant 2 in 1972 and 1973.
29	1.8	At plant 2 list the operating cost incurred in 1972 and 1973.
76	4.1	For depreciation management salary and interest cost list the amounts incurred in 1972 and 1973.
1	0.4	What was the operating cost at each plant?
72	4.0	What was the percent change in operating cost at each plant from 1972 to 1973?
2	2.1	In 1973 what percentage of the direct manufacturing cost was accounted for by operating cost?
69	2.4	What was the change in total manufacturing cost from 1972 to 1973?
9	1.2	What was the percent change in total revenues from 1972 to 1973?
2	2.2	In 1973 what percentage of the direct manufacturing cost was accounted for by operating cost?
69	2.4	What was the change in total manufacturing cost from 1972 to 1973?
69	3.4	What was the percent change in total manufacturing cost from 1972 to 1973?
9	1.2	What was the percent change in total revenues from 1972 to 1973?
26	3.3	What was the percent change in total overhead costs from 1972 to 1973?
22	1.7	Define P-cost to be the sum of overhead cost and manufacturing cost.
6	1.0	What percentage of the P-cost is accounted for by overhead cost?
5	0.4	For what year was that figure?
4	0.3	Give me the same figure for 1972.
2	0.1	How is profit computed?
4	0.2	How is total cost computed?
1	0.4	Are production cost and manufacturing cost the same?
1	0.0	Hello!
0	1.7	Give me two tables, the contribution margin for all products in each plant for the years 1972 and 1973.
2	0.4	Give me the total sales for 1972 and 1973.
5	0.5	Give me the sales volume by product for the years 1972 and 1973.
33	4.7	Give me the following proportions: the sales of products one, two and five divided by the total sales for 1972 and 1973.
13	1.3	Give me the average costs and the budgeted costs for the five products for 1973 and 1972.
2	0.0	Unit costs.
9	0.5	Give me the distribution of the sales of product four by plant.
7	0.7	Distribution of the sales of product 4 by plant for the year 1972.
2	0.2	Give me the budget for plant 4.
3	0.5	Give me the direct costs and the overheads for 1972 and 1973.
7	1.1	Was the actual overhead expense in plant 4 higher than the budgeted amount in 1973?
1	0.1	By how much?
128	16.1	Suppose the sales in 1973 had remained unchanged, would the profit picture have altered if the selling price of product 1 had been increased to allow a profit margin of $5.5, and by how much?
1	0.4	Next, would the sales have altered significantly if there had been this price increase?
28	1.5	Even though the plants are not operated as profit centers, could you tell me the contribution to profits from each plant for the years 1972 and 1973?
9	0.7	Give me the sales by product for plant two for the years 1972 and 1973.
5	0.3	Give me the proportional increase in the sales of the various products.
5	0.6	Give me the prices for the various products for the last two years.
1	0.0	No.
2	0.5	Please give me the sales for 1969 70 71 72 and 73.
3	0.7	Total profit margin for 69 70 71 72 and 73.
2	0.1	Total profit.
1	0.2	Profit margins for each product?
4	0.3	Sales form each plant during 73.
3	0.3	Sales from each plant during 72.
12	1.8	The ratio of products costing $6.25 and $5.00 from each plant during 72 and 73.
5	0.5	Can you give me data on product mix from each plant?
7	1.4	Give me the overhead costs from each plant during 72 and 73.
39	2.4	Give me the ratios of overhead costs and sales from each plant for 72 and 73.
24	2.1	Give me the ratios of overhead costs and sales for plants 1 2 3 4 for 72 and 73.
1	0.1	For 72 and 73.
133	3.8	Give ratios of manufacturing costs to sales for plants 1 2 3 and 4 for 72 and 73.
37	1.1	Give percentage change in sales for each plant for years 72 and 73.
63	1.8	Give percentage change in overhead costs for all plants for years 72 and 73.
1	0.3	What is total revenue for company?
5	0.3	What was the cost of goods sold?

2	0.1	I want the sum.
1	0.2	What was the net income?
2	0.6	What is the cost for each product in each plant?
1	0.1	Unit cost.
9	1.8	What was the actual unit cost change per product in 1973 over 1972?
30	1.5	How much did overhead expense increase in 1973 over 1972 in each plant?
1	0.1	What is plant 0?
2	0.2	Will our customers pay more for the product?
1	0.0	Cancel.
9	0.9	What was the volume increase per product in 1973 over 1972?
9	0.9	What was overhead increase per location in 1973 over 1972?
2	0.3	Who are my customers and what are their volumes per customer?
1	0.3	What is the price of each product?
2	0.2	Display for 1972 and 1973.
6	1.9	Sales, overhead, selling price, overhead, direct manufacturing cost, and profit margin for all types.
1	0.3	Remember this request (call it request A).
2	0.1	Can you format reports?
2	0.4	Please respond to request A for years 1972 and 1973.
2	0.1	Display sales.
7	0.8	Display sales for years 1972 and 1973 by battery types.
1	0.2	Call chas the ratio (overhead/sales).
1	0.0	Congratulations.
2	0.4	Please retain the results of specifications until I change them.
6	0.7	Display for years 1972 and 1973 sales and chas by battery type.
6	0.6	Display ((sales in 1972 - sales in 1973)/sales in 1972).
1	0.2	Remember to retain specifications of previous requests.
1	0.2	Call last displayed quantity "sales growth".
7	0.5	Display sales growth for all types.
3	0.4	Display average cost for 1972 and 1973.
4	0.4	Production cost averaged over sales.
1	0.0	Again by product please.
20	0.7	Display cost of goods sold for product 1.
1	0.4	What is the difference between "production cost" and "direct manufacturing cost"?
1	0.0	No they aren't.
2	0.1	Give me definition of margin.
1	0.0	Standard cost?
5	0.5	Let scvar be difference between standard costs and production costs.
4	0.6	Display scvar and sales growth for 1972 and 1973.
1	0.0	Cancel.
5	0.7	Display scvar for all products and all years.
2	0.1	What are my expense categories?
2	0.1	Display overhead.
8	0.6	Let alloc be ((overhead/production cost)*total production cost) for each product.
3	0.4	What data do you have regarding overhead expenditures?
3	0.5	What data do you have regarding production cost?
3	0.5	What data do you have regarding product mix?
9	0.7	Do you have production cost per unit for each type of product?
7	0.6	Print production cost per unit for product 1.
3	0.5	Print list price for product 1.
14	0.9	Print total manufacturing cost for product 1.
28	1.4	Print total manufacturing cost per unit of product 1.
5	0.6	Print overhead cost per unit of product 1.
16	1.2	What was the average budgeted cost per unit of product 1?
4	0.5	What does the average budgeted cost per unit include?
17	0.9	Print budgeted cost per unit of products 2, 3, 4.
7	0.9	Print direct production costs per unit for all products.
11	0.9	Print list prices per unit for all products.
2	0.8	What was the expected contribution margin for all products per unit?
2	0.9	What was the actual contribution margin for all products per unit?
7	1.5	What was the average selling price per unit for all products?
2	0.2	What were expected overhead costs?
1	0.2	What was actual overhead cost?
1	0.2	What was the planned product mix?
1	0.2	What was actual product mix?
23	1.4	Print production costs per unit and per plant for all products.
6	0.7	Do you have a list of overhead cost for each plant separately?
1	0.1	Print this list.
1	0.1	What is plant 0?
54	1.6	Do you have a list of production cost itemized per type of direct costs?
1	0.2	What was the budgeted direct material cost?
1	0.2	What was the direct material cost?
1	0.1	What was labor cost?
1	0.1	What was transportation cost?
1	0.3	What was material cost in 1972?

1	0.3	What was labor cost in 1972?
1	0.3	What was transportation cost in 1972?
16	1.0	Do you have records on sales per major customer in 1972 and 1973?
2	0.1	List data available.
3	0.8	Print the unit cost for battery type 1 at each plant.
27	1.5	List actual and budgeted unit costs for product 1 for 65 to 73.
32	1.5	List the data for the last 5 years for each product by unit cost.
192	1.7	Define equation discount (x) = (list price (x)-selling price (x))/(list price (x)).
1	0.2	Solve discount (product 1).
2	0.3	Print discount (product 1).
2	0.5	Solve discount (product 2), then print discount (product 2).
2	0.4	Solve discount (product 3), then print answer.
2	0.4	Solve discount (product 4), print answer.
2	0.4	Solve discount (product 5), print answer.
10	1.0	Print profit margin for each product for 72 and 73.
32	0.9	Define %sales(x) = (total sales product (x))/(total company sales).
7	0.7	Solve %sales(x) for each product for 72 and 73.
39	1.2	Print the number of units of each product produced by plant.
3	0.4	Print total sales volume by plant.
3	0.3	List profit margins by plant.
5	0.4	List production costs by plant.
5	0.4	List overhead costs by plant.
8	0.9	Define %CHoverhead(T) = (overhead(T) - overhead(T-1))/(overhead(T-1)).
2	0.2	Why are there 5 plants?
2	0.6	What were the major increases in overhead in plant 1?
14	0.6	Give dollar figures for overhead expenses for plant 1.
2	0.3	Itemize overhead costs for plant 1.
8	0.8	Define %CH (item T) = (item(T)-item(T-1))/(item(T-1)).
1	0.2	Let item be depreciation, and T be 73.
1	0.2	Print the last answer.
3	0.2	Let item be operating cost.
1	0.1	Let management salaries be item.
1	0.2	Let item be interest cost.
8	0.5	Let item be operating cost by plant.
6	0.3	What makes up operating costs?
1	0.1	Let item be *entertainment expenses*.
3	0.4	Print for total, and each plant.
2	0.2	Let item be interest cost by plant.
1	0.3	What were the overhead expenses in 1973?
3	1.0	What was the percentage increase in overhead cost, 1973 vs 1972?
14	1.6	What was the percentage increase in freight and distribution costs for the same period?
7	1.3	What was the actual value of freight and distribution costs in 1973?
2	0.4	Was there an increase in truckers fees in 1973?
52	1.4	Are all increases from freight carriers passed on to the customer?
11	0.4	Is transportation cost included in overhead?
3	0.8	What were the sales by product (5 products) for 1972 and 1973?
2	0.6	What was the turnover by product for 1972 and 1973?
56	1.4	Divide cost of sales by average inventory for each year for each product and give us the result.
1	0.1	For 1972 and 1973.
3	1.0	What was the profit margin for each product for 1972 and 1973?
7	1.8	What was the percentage of total sales for each product for 1972 and 1973?
3	0.6	What cost items are included in overhead cost?
6	1.0	What were the overhead costs for 1972 and 1973 for each plant?
30	1.9	Can you give the percent of total overhead cost of each plant for 1972 and 1973?
4	1.0	What was the percent change 1972 vs 1973 for each plant?
8	0.5	Do you have a model for measuring customer service?
77	1.4	Do you have a count of the number of sales requests and the number of requests filled?
1	0.3	What types of data do you have?
4	0.2	Is region recorded by product?
4	0.2	Is revenue recorded by product?
3	0.3	What are revenues for each product?
3	0.2	What are sales by plant?
7	0.4	What are sales by plant by product?
132	1.4	Can you subtract 1972 sales by plant by product from 1973 sales by plant by product?
132	1.3	Subtract 1972 sales by plant by product from 1973 sales by plant by product.
3	0.6	Did any product costs exceed budget in 73?
22	0.7	By plant by product which costs exceeded budget?
1	0.5	Which product of the five had the largest percentage variance?
2	0.3	In 1972 which product or products had largest variances?
1	0.5	What were 1972 and 1973 profit margins by product?
10	0.3	Can you give unit costs by plant by product?
2	0.8	What were actual costs per unit for plant two?
1	0.2	What were unit costs for 1972?
2	0.5	What was product mix by percent in 72?

2	0.5	What was product mix by percent in 73?
15	0.3	Were prices raised in 1973 over 1972?
2	.6	What were 1972 and 1973 prices for each product?

Appendix II - Results with the LADDER-TODS Collection

2	0.3	What kind of information do you know about?
2	0.3	Is there a doctor on board the Biddle?
3	0.4	Display all the American cruisers in the North Atlantic.
6	0.7	What is the name and location of the carrier nearest to New York?
1	0.3	What is the commanding officer's name?
1	0.1	Who commands the Kennedy?
1	0.2	What is the Kennedy's beam?
1	0.2	When will the Los Angeles reach Norfolk?
5	0.5	Tell me when Taru is scheduled to leave port.
4	0.2	Where is she scheduled to go?
2	0.4	When will Los Angeles arrive in its home port?
2	0.2	When will the Sturgeon arrive on station?
5	0.8	What aircraft units are embarked on the Constellation?
8	0.7	To which task organization is Knox assigned?
1	0.1	Where is the Sellers?
1	0.0	Where is Luanda?
4	0.5	What is the next port of call of the Santa Inez?
2	0.1	When will Tarifa get underway?
1	0.6	Which convoy escorts have inoperative sonar systems?
1	0.1	When will they be repaired?
2	1.8	Which U.S. Navy DDGs have casreps involving radar systems?
1	0.3	What Soviet ship has hull number 855?
2	0.4	To what class does the Soviet ship Minsk belong?
2	0.3	What class does the Whale belong to?
19	2.0	What is the normal steaming time for the Wainwright from Gibraltar to Norfolk?
3	0.5	What American ships are carrying vanadium ore?
1	0.1	How far is it to Norfolk?
1	0.1	How far away is Norfolk?
1	0.2	How many nautical miles is it to Norfolk?
2	0.3	How many miles is it to Norfolk from here?
2	0.3	How close is the Baton Rouge to Norfolk?
2	0.2	How far is the Adams from the Aspro?
2	0.3	What is the distance from Gibraltar to Norfolk?
1	0.2	What is the nearest oiler?
1	0.3	What is the nearest oiler to the Constellation?
3	0.4	How far is it from Naples to 23-00N, 45-00W?
2	0.5	What is the distance from the Kitty Hawk to Naples?
4	0.6	How long would it take the Independence to reach 35-00N, 20-00W?
1	0.1	How long is the Philadelphia?
4	0.3	How long would it take the Aspro to join Kennedy?
28	1.3	What is the nearest ship to Naples with a doctor on board?
2	1.4	What is the nearest USN ship to the Enterprise with an operational air search radar?
1	0.2	What is known about that ship?
2	0.7	How many merchant ships are within 400 miles of the Hepburn?
4	0.3	What are their identities and last reported locations?
1	0.2	What cargo does the Pecos have?
1	0.1	Who is CTG67.3?
3	0.6	What are the length, width, and draft of the Kitty Hawk?
6	0.5	To whom is the Harry E. Yarnell attached?
2	0.7	What type ships are in the Knox class?
1	0.4	Where are the Charles F. Adams class ships?
2	0.1	What are their current assignments?
2	1.0	What subs in the South Atlantic are within 1000 miles of the Sunfish?
2	0.2	What is the Kittyhawk doing?
1	0.9	How many USN asw capable ships are in the Med?
1	0.0	Where are they?
4	0.4	What are their current assignments and fuel states?
4	1.0	What ships are NOT at combat readiness rating C1?
6	0.4	When will Reeves achieve readiness rating C1?
8	0.4	Why is Hoel at readiness rating C2?
1	0.2	When will the sonar be repaired on the Sterett?
5	0.8	What ships are carrying cargo for the United States?
1	0.1	Where are they going?
1	0.1	What are they carrying?
2	0.0	When will they arrive?
2	0.2	Where is Gridley bound?
14	1.3	Which cruisers have less than 50 per cent fuel on board?
1	0.1	Where are all the merchant ships?
2	0.3	When will the Kitty Hawk's radar be up?
1	0.4	What ships are in the Los Angeles class?
1	0.3	What command does Adm. William have?

3	0.3	Under whose opcon is the Dale?
1	0.2	Show me where the Kennedy is!
1	0.3	What ship has hull number 148?
4	0.7	What is the next port of call for the South Carolina?
4	0.2	Are doctors embarked in the Kawishiwi?
2	0.4	What kind of cargo does the Francis McGraw have?
5	0.9	What air group is embarked in the Constellation?
2	0.5	What do you know about the employment schedule of the Lang?
2	0.5	Which systems are down on the Kitty Hawk?
5	0.5	What ships in the Med have doctors embarked?
2	1.2	How many ships carrying oil are within 340 miles of Mayport?
4	1.2	What sub contacts are within 300 miles of the Enterprise?
9	2.0	List the current position and heading of the US Navy ships in the Mediterranean every 4 hours.
2	0.6	What is the status of the Enterprise's air search radar?
1	0.2	Where is convoy NL53 going?
2	0.3	What convoy is the Transgermania in?
2	0.4	How many embarked units are in Constellation?
1	0.3	What ships are in British ports?
4	1.4	What U.S. ships are within 500 miles of Wilmington?
2	1.2	What U.S. ships faster than the Gridley are in Norfolk?
1	0.4	What is the fastest ship in the Mediterranean Sea?
2	0.3	How close is that ship to Naples?
1	0.1	What is its home port?
2	0.4	Print the American cruisers' current positions and states of readiness!
2	0.2	How is the Los Angeles powered?
11	4.1	What ship having a normal cruising speed greater than 30 knots is the largest?
30	1.6	Display the last reported position of all ships that are in the North Atlantic.
1	0.3	When did the Endeavour depart the port of New York?
1	0.8	What nationality is the ship with international radio call sign UA1D?
1	0.3	What ports are in the database?
2	0.8	What merchant ships are enroute to New York and within 500 miles of the Saratoga?
2	0.4	To what country does the fastest sub belong?

Appendix III - How to Understand the ATN Charts

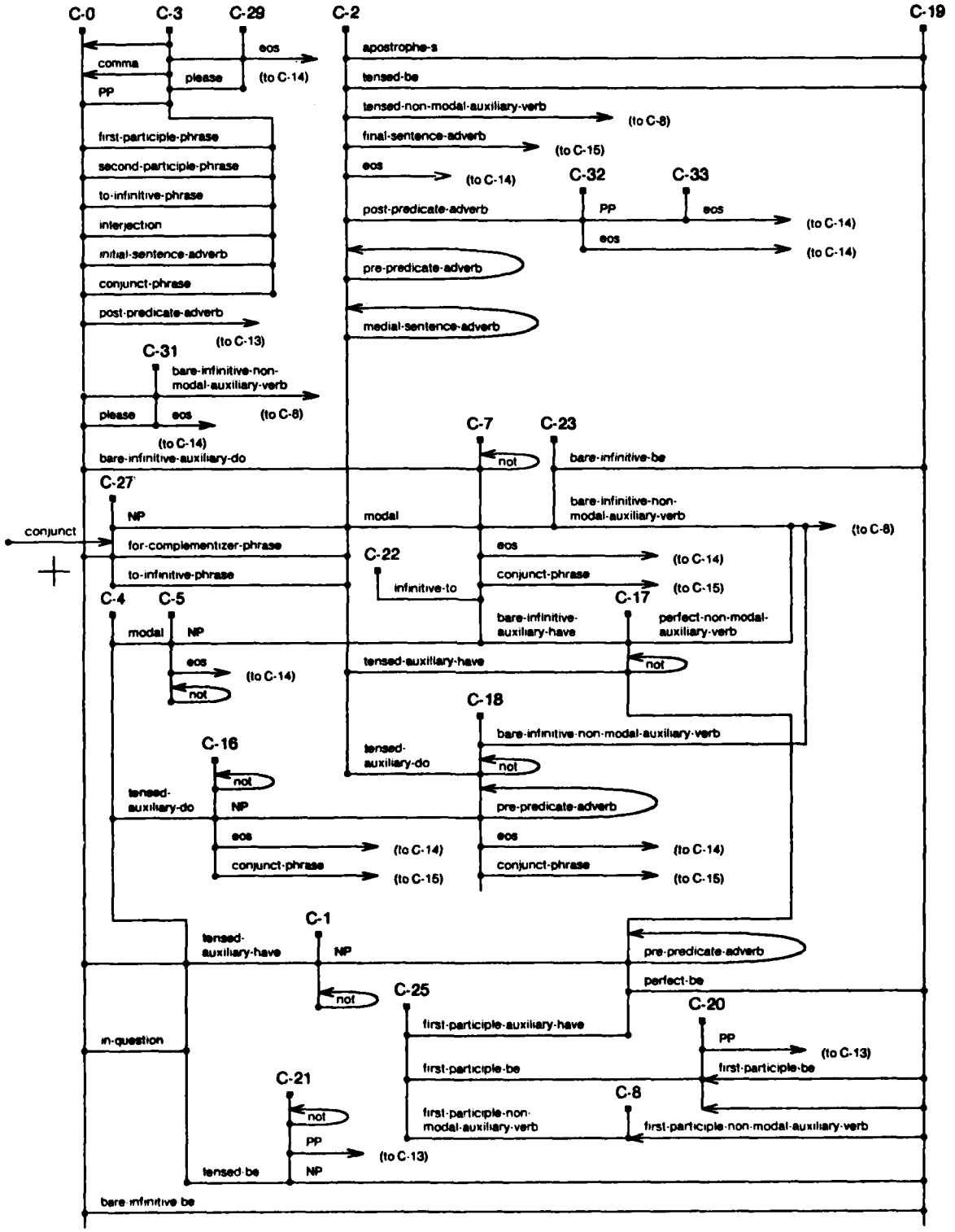
The ATN Charts used in EQSP immediately follow this page. These charts only give the context-free rules; the augmentations have been omitted except in a few cases. When two context-free rules have the same left side, they have been merged into regular expressions. These new rules have the same recognition power as the original rules, but not the same power to generate syntactic structures. For example, the first rules would find the two expressions (137)-(138) while the second will find only (139).

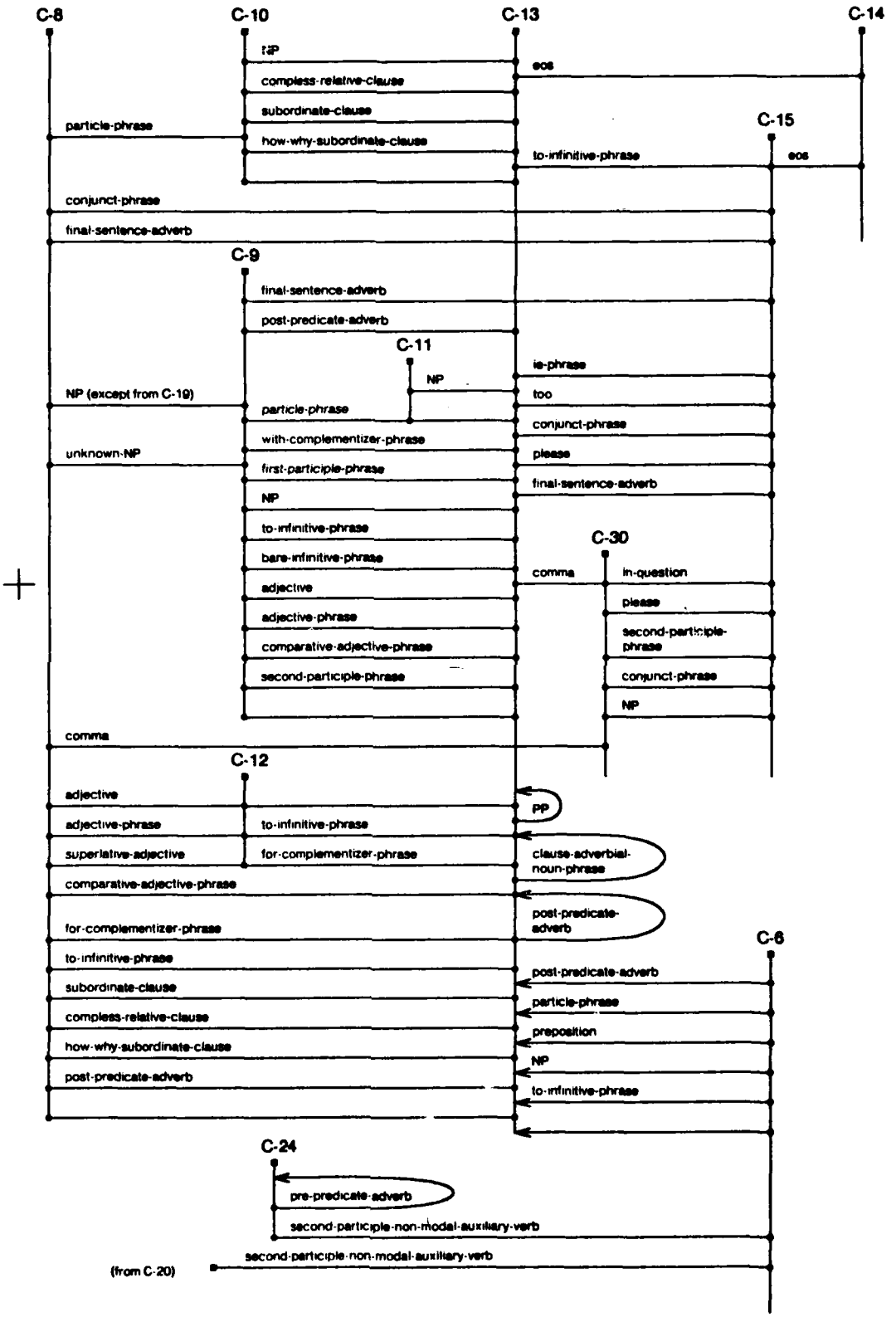
(137) $[_{NP}\ [_{N}$ fire $[_{NP}\ [_{N}$ plug$]\ [_{N}$ hose$]]]]$

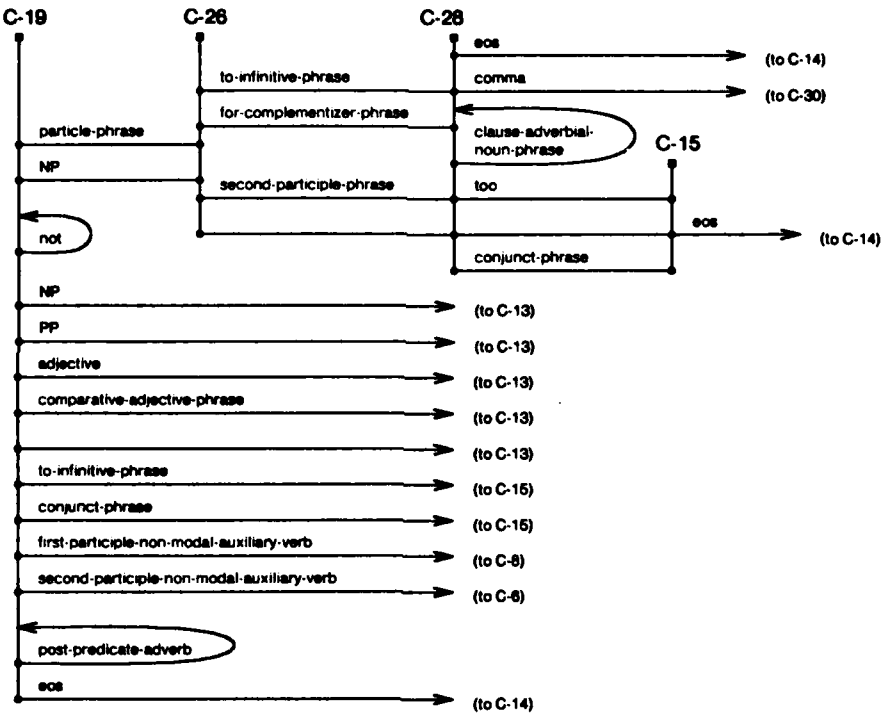
(138) $[_{NP}\ [_{NP}\ [_{N}$ fire$]\ [_{N}$ plug$]]\ [_{N}$ hose$]]$

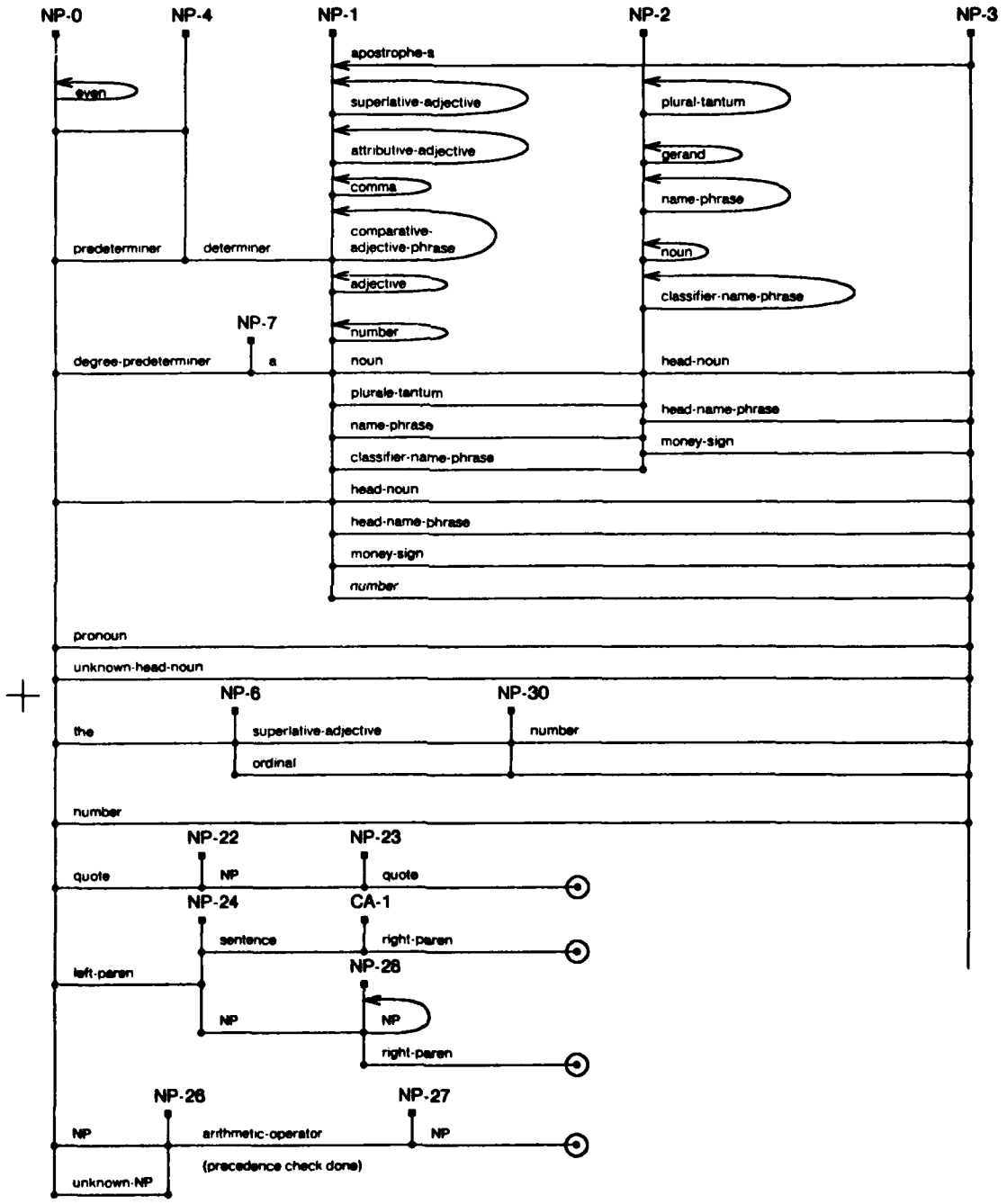
(139) $[_{NP}\ [_{NP}$ fire$]\ [_{NP}$ plug$]\ [_{NP}$ hose$]]$

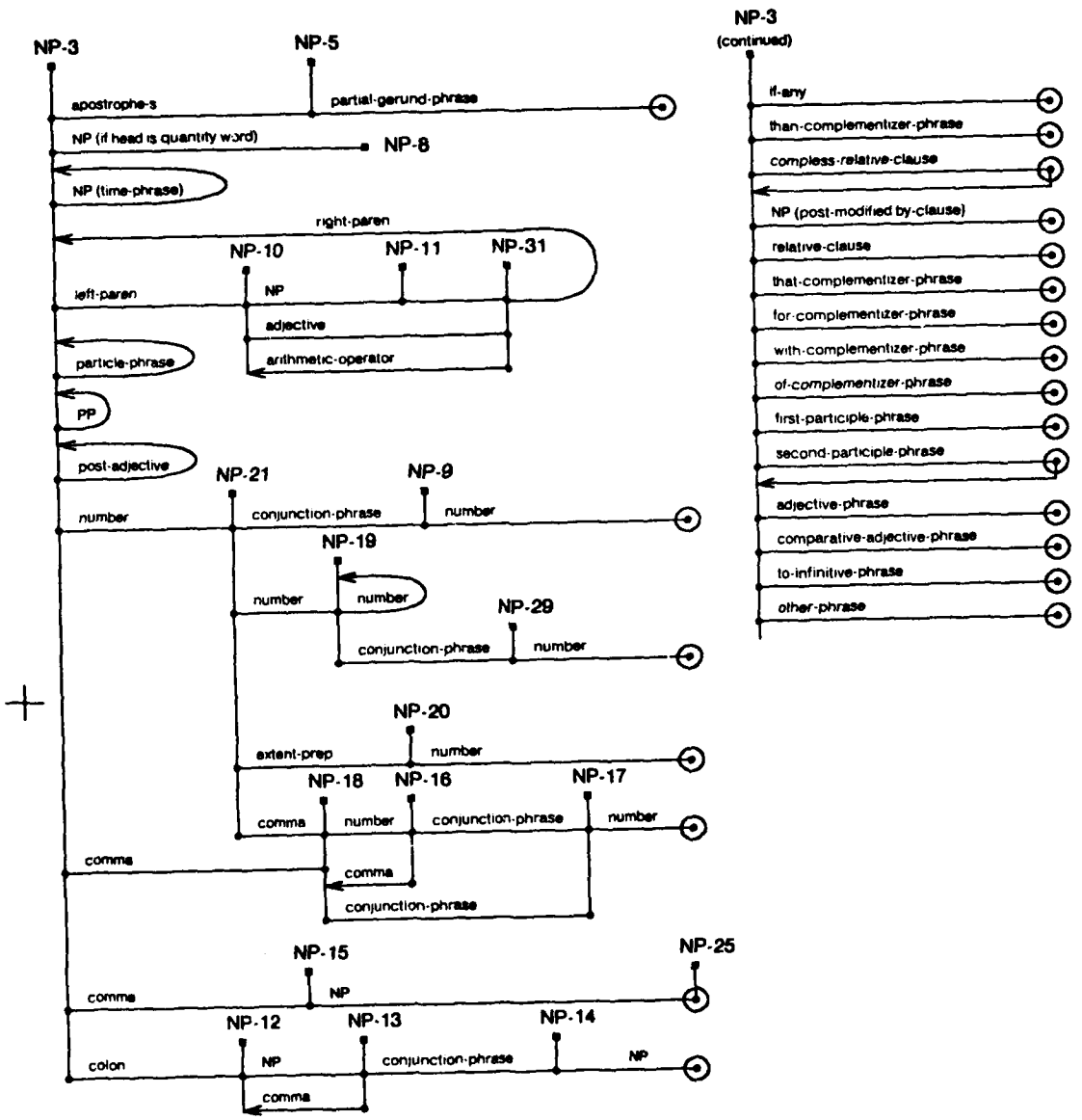
Nodes have been elongated into bars so that there is enough room for the arcs. Labeled arcs can be traversed when that part of speech is found (conditional on the augmentations). Unlabeled links are jump arcs and can be traversed at any time. If a node has an arrow at its left, it is traversed from right to left, otherwise it is traversed left to right.

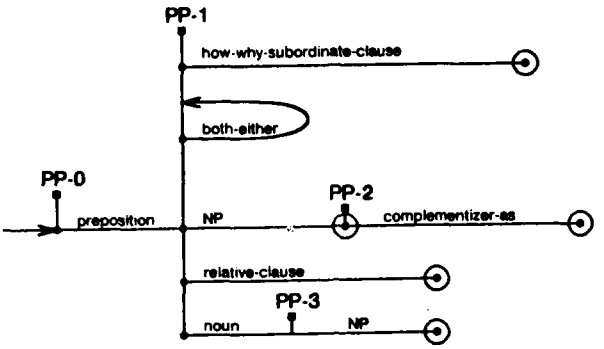

C-0
C-3
C-29
C-2
C-19
eos
comma
please
(to C-14)
PP
apostrophe-s
tensed-be
tensed-non-modal-auxiliary-verb
(to C-8)
final-sentence-adverb
(to C-15)
first-participle-phrase
second-participle-phrase
to-infinitive-phrase
interjection
initial-sentence-adverb
conjunct-phrase
post-predicate-adverb
(to C-13)
eos
(to C-14)
C-32
C-33
post-predicate-adverb
PP
eos
(to C-14)
eos
(to C-14)
pre-predicate-adverb
medial-sentence-adverb
C-31
bare-infinitive-non-modal-auxiliary-verb
please
eos
(to C-8)
(to C-14)
bare-infinitive-auxiliary-do
C-7
C-23
not
bare-infinitive-be
C-27
NP
modal
bare-infinitive-non-modal-auxiliary-verb
(to C-8)
conjunct
for-complementizer-phrase
to-infinitive-phrase
C-22
infinitive-to
eos
(to C-14)
conjunct-phrase
(to C-15)
C-4
C-5
C-17
modal
NP
bare-infinitive-auxiliary-have
perfect-non-modal-auxiliary-verb
eos
(to C-14)
not
tensed-auxiliary-have
not
C-18
bare-infinitive-non-modal-auxiliary-verb
C-16
tensed-auxiliary-do
not
not
tensed-auxiliary-do
NP
pre-predicate-adverb
eos
(to C-14)
conjunct-phrase
(to C-15)
eos
(to C-14)
conjunct-phrase
(to C-15)
C-1
tensed-auxiliary-have
NP
pre-predicate-adverb
not
perfect-be
C-25
C-20
first-participle-auxiliary-have
in-question
PP
(to C-13)
first-participle-be
first-participle-be
C-21
C-8
not
first-participle-non-modal-auxiliary-verb
first-participle-non-modal-auxiliary-verb
PP
(to C-13)
tensed-be
NP
bare-infinitive-be

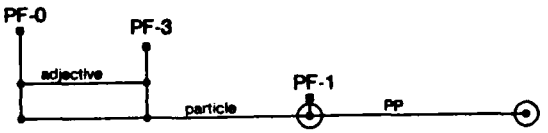

C-8
C-10
C-13
C-14
NP
compless-relative-clause
subordinate-clause
particle-phrase
how-why-subordinate-clause
eos
C-15
to-infinitive-phrase
eos
conjunct-phrase
final-sentence-adverb
C-9
final-sentence-adverb
post-predicate-adverb
C-11
NP
particle-phrase
NP (except from C-19)
with-complementizer-phrase
unknown-NP
first-participle-phrase
NP
to-infinitive-phrase
bare-infinitive-phrase
adjective
adjective-phrase
comparative-adjective-phrase
second-participle-phrase
ie-phrase
too
conjunct-phrase
please
final-sentence-adverb
C-30
comma
in-question
please
second-participle-phrase
conjunct-phrase
NP
comma
C-12
adjective
adjective-phrase
superlative-adjective
comparative-adjective-phrase
for-complementizer-phrase
to-infinitive-phrase
subordinate-clause
compless-relative-clause
how-why-subordinate-clause
post-predicate-adverb
PP
to-infinitive-phrase
for-complementizer-phrase
clause-adverbial-noun-phrase
post-predicate-adverb
C-6
post-predicate-adverb
particle-phrase
preposition
NP
to-infinitive-phrase
C-24
pre-predicate-adverb
second-participle-non-modal-auxiliary-verb
second-participle-non-modal-auxiliary-verb
(from C-20)

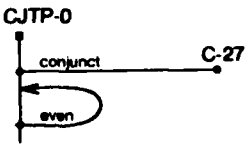

C-19
C-26
C-28
C-15
eos
(to C-14)
to-infinitive-phrase
comma
(to C-30)
for-complementizer-phrase
particle-phrase
clause-adverbial-noun-phrase
NP
second-participle-phrase
too
eos
(to C-14)
not
conjunct-phrase
NP
(to C-13)
PP
(to C-13)
adjective
(to C-13)
comparative-adjective-phrase
(to C-13)
(to C-13)
to-infinitive-phrase
(to C-15)
conjunct-phrase
(to C-15)
first-participle-non-modal-auxiliary-verb
(to C-6)
second-participle-non-modal-auxiliary-verb
(to C-6)
post-predicate-adverb
eos
(to C-14)

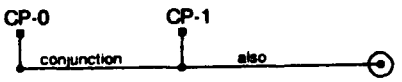

NP-0
NP-4
NP-1
NP-2
NP-3
apostrophe-s
even
superlative-adjective
plural-tantum
attributive-adjective
gerund
comma
name-phrase
predeterminer
determiner
comparative-adjective-phrase
noun
adjective
classifier-name-phrase
NP-7
number
degree-predeterminer
a
noun
head-noun
plurale-tantum
head-name-phrase
name-phrase
money-sign
classifier-name-phrase
head-noun
head-name-phrase
money-sign
number
pronoun
unknown-head-noun
NP-6
NP-30
the
superlative-adjective
number
ordinal
number
NP-22
NP-23
quote
NP
quote
NP-24
CA-1
sentence
right-paren
NP-28
left-paren
NP
NP
right-paren
NP-26
NP-27
NP
arithmetic-operator
NP
(precedence check done)
unknown-NP

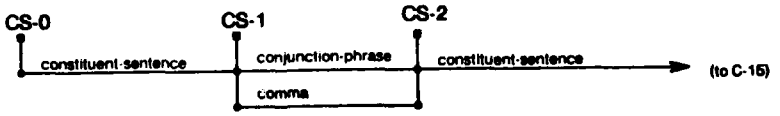

NP-3
NP-5
apostrophe-s
partial-gerund-phrase
NP (if head is quantity word)
NP-8
NP (time-phrase)
right-paren
NP-10
NP-11
NP-31
left-paren
NP
adjective
arithmetic-operator
particle-phrase
PP
post-adjective
NP-21
NP-9
number
conjunction-phrase
number
NP-19
number
number
NP-29
conjunction-phrase
number
NP-20
extent-prep
number
NP-18
NP-16
NP-17
comma
number
conjunction-phrase
number
comma
comma
conjunction-phrase
NP-15
NP-25
comma
NP
NP-12
NP-13
NP-14
colon
NP
conjunction-phrase
NP
comma
NP-3
(continued)
if-any
than-complementizer-phrase
compless-relative-clause
NP (post-modified by-clause)
relative-clause
that-complementizer-phrase
for-complementizer-phrase
with-complementizer-phrase
of-complementizer-phrase
first-participle-phrase
second-participle-phrase
adjective-phrase
comparative-adjective-phrase
to-infinitive-phrase
other-phrase

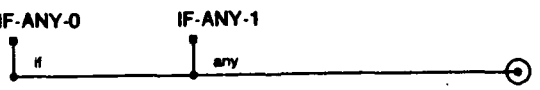

PP-1
how-why-subordinate-clause
both-either
PP-0
preposition
NP
PP-2
complementizer-as
relative-clause
PP-3
noun
NP

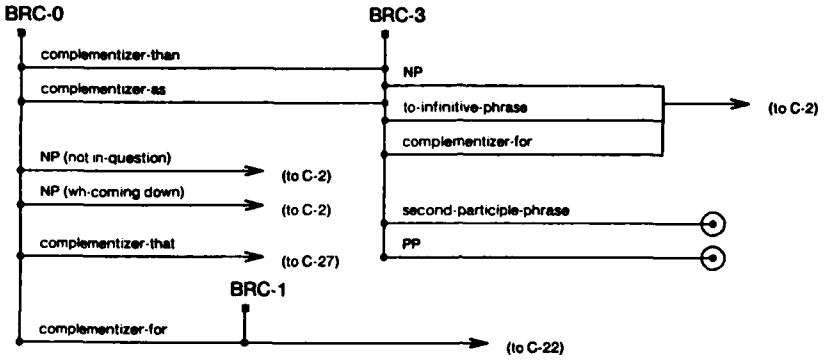

PF-0
PF-3
adjective
particle
PF-1
PP

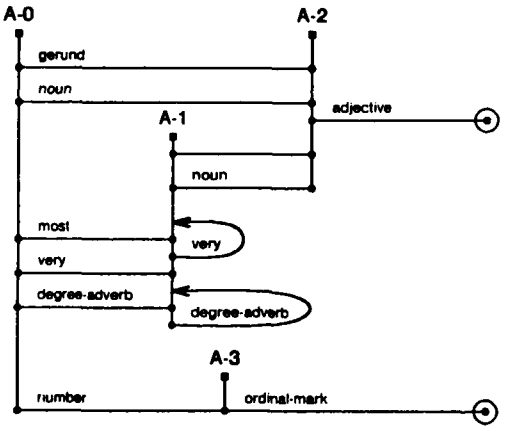

CJTP-0
conjunct
C-27
even

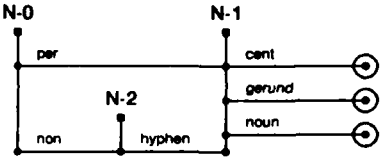

CP-0
CP-1
conjunction
also

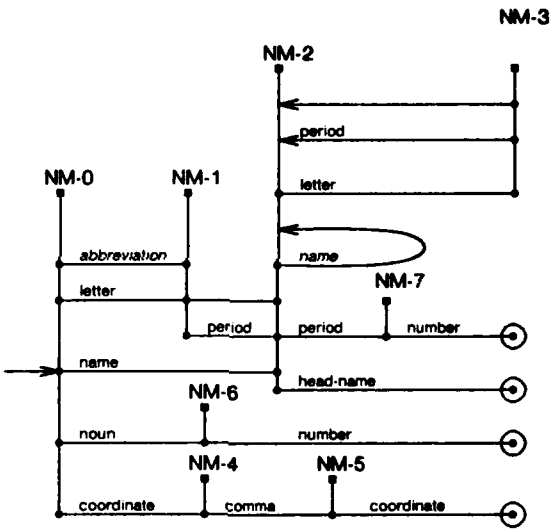

CS-0
CS-1
CS-2
constituent-sentence
conjunction-phrase
constituent-sentence
(to C-15)
comma

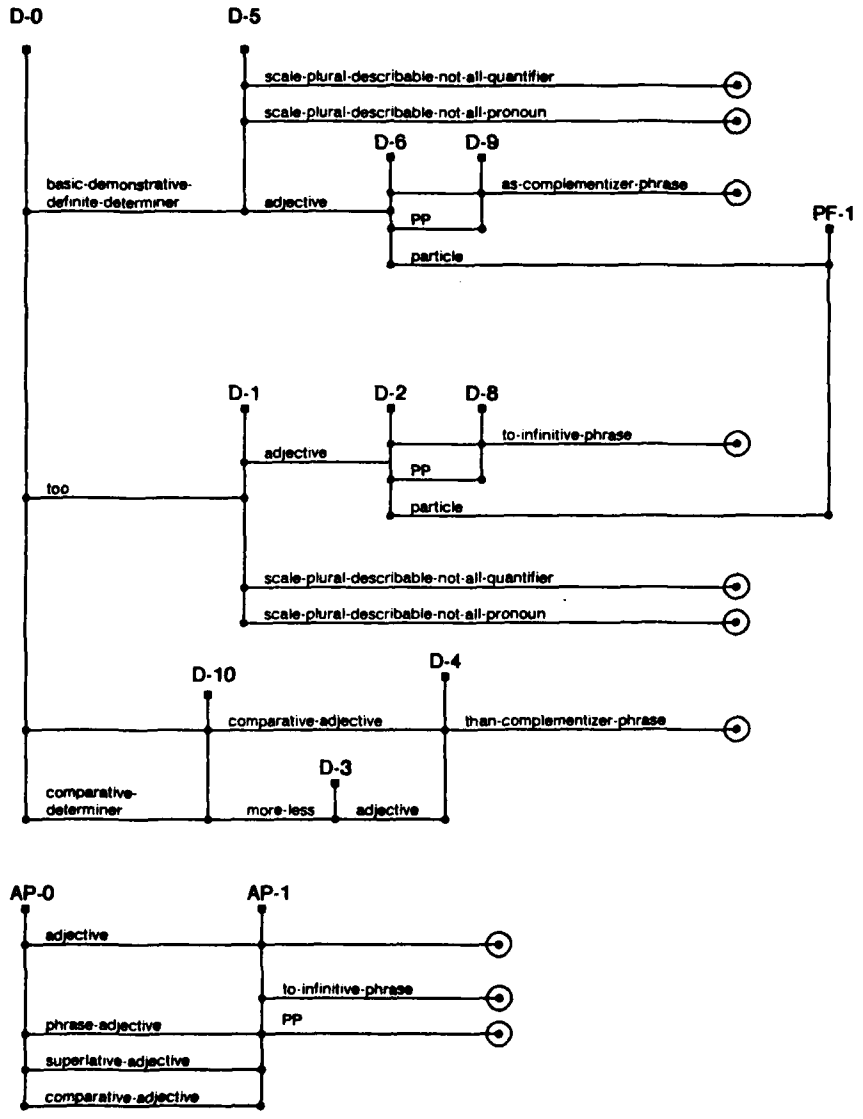

IF-ANY-0
IF-ANY-1
if
any

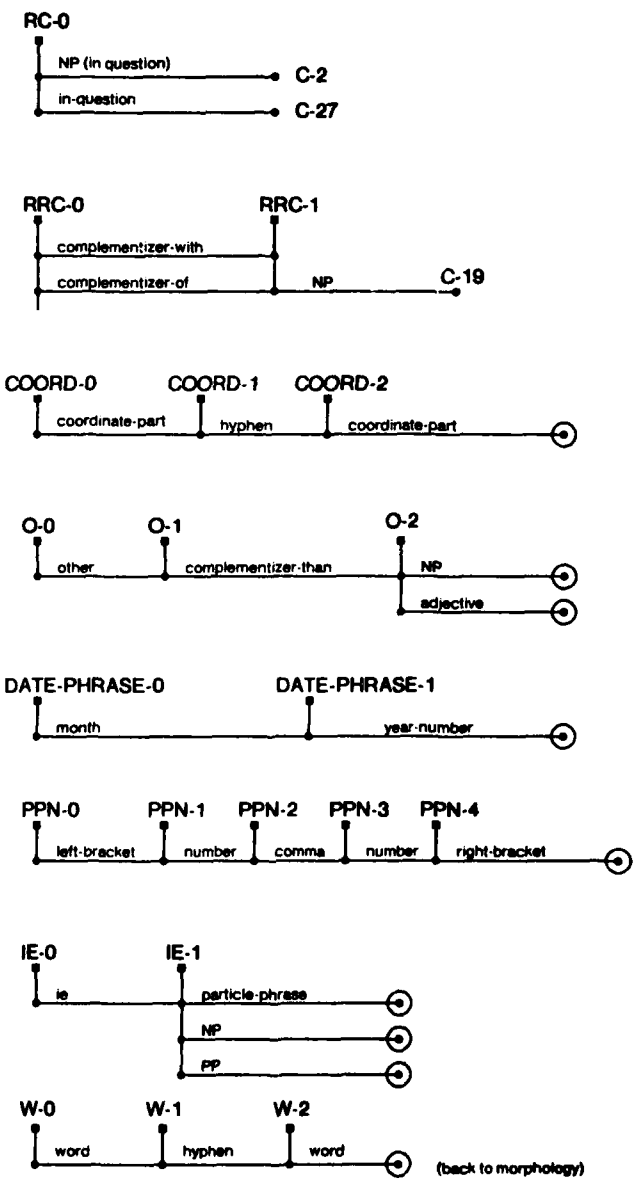

BRC-0
BRC-3
complementizer-than
complementizer-as
NP
to-infinitive-phrase
(to C-2)
complementizer-for
NP (not in-question)
(to C-2)
NP (wh-coming down)
(to C-2)
second-participle-phrase
PP
complementizer-that
(to C-27)
BRC-1
complementizer-for
(to C-22)


A-0
A-2
gerund
noun
adjective
A-1
noun
most
very
very
degree-adverb
degree-adverb
A-3
number
ordinal-mark


N-0
N-1
per
cent
gerund
N-2
noun
non
hyphen


NM-3
NM-2
period
NM-0
NM-1
letter
abbreviation
name
NM-7
letter
period
period
number
name
head-name
NM-6
noun
number
NM-4
NM-5
coordinate
comma
coordinate


D-0
D-5
scale-plural-describable-not-all-quantifier
scale-plural-describable-not-all-pronoun
D-6
D-9
basic-demonstrative-definite-determiner
adjective
as-complementizer-phrase
PP
PF-1
particle
D-1
D-2
D-8
adjective
to-infinitive-phrase
PP
too
particle
scale-plural-describable-not-all-quantifier
scale-plural-describable-not-all-pronoun
D-10
D-4
comparative-adjective
than-complementizer-phrase
D-3
comparative-determiner
more-less
adjective
AP-0
AP-1
adjective
to-infinitive-phrase
PP
phrase-adjective
superlative-adjective
comparative-adjective


RC-0
NP (in question)
C-2
in-question
C-27
RRC-0
RRC-1
complementizer-with
complementizer-of
NP
C-19
COORD-0
COORD-1
COORD-2
coordinate-part
hyphen
coordinate-part
O-0
O-1
O-2
other
complementizer-than
NP
adjective
DATE-PHRASE-0
DATE-PHRASE-1
month
year-number
PPN-0
PPN-1
PPN-2
PPN-3
PPN-4
left-bracket
number
comma
number
right-bracket
IE-0
IE-1
ie
particle-phrase
NP
PP
W-0
W-1
W-2
word
hyphen
word
(back to morphology)

OFFICIAL DISTRIBUTION LIST

Director — 2 copies
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Program Management

Office of Naval Research — 3 copies
800 North Quincy Street
Arlington, Virginia 22217

Attention: Marvin Denicoff, Code 437

Office of Naval Research — 1 copy
Resident Representative
Massachusetts Institute of Technology
Building E19-628
Cambridge, Mass. 02139

Attention: A. Forrester

Director — 6 copies
Naval Research Laboratory
Washington, D.C. 20375

Attention: Code 2627

Defense Technical Information Center — 12 copies
Cameron Station
Arlington, Virginia 22314

Office of Naval Research — 1 copy
Branch Office/Boston
Building 114, Section D
666 Summer Street
Boston, Mass. 02210